AD-A044 612 HONEYWELL INFORMATION SYSTEMS INC MCLEAN VA FEDERAL --ETC F/G 9/2
TRANSACTION PROCESSING OPERATING SYSTEM (TPOS).(U)
AUG 77 R EWING, J BIELSKI F30602-76-C-0277
UNCLASSIFIED RADC-TR-77-276 NL

1 OF 3
ADA044612

12

RADC-TR-77-276
Final Technical Report
August 1977

TRANSACTION PROCESSING OPERATING SYSTEM (TPOS)

Honeywell Information Systems Inc.

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

APPROVED: Raymond A. Liuzzi

RAYMOND A. LIUZZI
Project Engineer

APPROVED: [illegible]

ROBERT D. KRUTZ, Colonel, USAF
Chief, Information Sciences Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(19) **REPORT DOCUMENTATION PAGE**	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER (18) RADC-TR-77-276 / 2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER (9)
4. TITLE (and Subtitle) (6) TRANSACTION PROCESSING OPERATING SYSTEM (TPOS).	5. TYPE OF REPORT & PERIOD COVERED Final Technical Report. Jul 76 - Jul 77,
	6. PERFORMING ORG. REPORT NUMBER N/A
7. AUTHOR(s) (10) Ronald Ewing John Bielski	8. CONTRACT OR GRANT NUMBER(s) (15) F30602-76-C-0277
9. PERFORMING ORGANIZATION NAME AND ADDRESS Honeywell Information Systems Inc. Federal Systems Operations 7900 West Park Drive, McLean VA 22101	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS (16) 55810274 62702F (17) 02
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (ISIM) Griffiss AFB NY 13441	12. REPORT DATE (11) August 1977
	13. NUMBER OF PAGES 261
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Same (12) 260p.	15. SECURITY CLASS. (of this report) UNCLASSIFIED
	15a. DECLASSIFICATION DOWNGRADING SCHEDULE N/A
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.	
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) Same	
18. SUPPLEMENTARY NOTES RADC Project Engineer: Raymond A. Liuzzi (ISIM)	
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Transaction Processing; Operating Systems; Computers; Data Processing	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The Transaction Processing Operating System (TPOS) is a specialized system designed to control a subset of the total computer environment, on the Honeywell H6000 architecture, of Transaction Processing Application Programs (TPAPS).

The TPOS system is designed to operate under Release 2/H of the General Comprehensive Operating Supervisor (GCOS), the General Remote Terminal Supervisor (GRTS) on the Network Processing Supervisor (NPS) and in →

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

409 690 LB

conjunction with the Transaction Processing System (TPS).

This report includes the user, site and program logic documentation. In addition, this report indicates specific usage of a monitoring and a debugging capability provided within the TPOS environment.

DOCUMENTATION PREFACE

Computer Definition

The Transaction Processing Operating System is designed to execute within a Honeywell Series 6000 or Series 60/Level 66 Information System, with any allowable memory size. One DATANET 30, 305, 355, or 6600 or 355 Communications Processor is required to perform front-end processing functions and at least one of the following terminals is required: Teletype Models 33, 35, 37, or 38; GE TermiNet 300; VIP Series 765, 775, 785, 7700 keyboard/display terminals.

System Definition

The Transaction Processing Operating System (TPOS) is designed to operate under Release 2/H of the General Comprehensive Operating Supervisor (GCOS), the General Remote Terminal Supervisor (GRTS) or the Network Processing Supervisor (NPS) and in conjunction with the Transaction Processing System (TPS).

CONTENTS

CONTENTS (cont)

CONTENTS (cont)

CONTENTS (cont)

CONTENTS (cont)

EVALUATION

This effort has provided a number of important enhancements designed to improve the operational capability of the Transaction Processing Operating System (TPOS) which executes under the GCOS Operating System on the H6000 Computer System.

The establishment of a sophisticated programming environment for command and control applications is a criteria defined under TPO V 3.5. Within this environment, a requirement exists for real-time processing on the H6000 computer architecture. The TPOS system provides a vehicle to process in a real-time mode a number of transactions to a wide variety of previously defined data structures. The TPOS system also provides the following capabilities:

. Ability to tailor the environment to the specific transaction processing needs of a user.

. Reduction in the number of programs dedicated to a user's site transaction processing requirements.

. Multiple concurrent execution of the same Transaction Processing Application Programs.

The capability provided by the TPOS system can be immediately utilized by a wide variety of current applications which execute under control of the H6000 Transaction Processor. It can provide a means of defining a real-time programming environment within large scale computer architectures.

Raymond A. Liuzzi

RAYMOND A. LIUZZI
Project Engineer

PURPOSE

The purpose of the Transaction Processing Operating System is to:

(1) Allow multiple copies of the same Transaction Processing Applications Program (TPAP) to execute concurrently,

(2) Increase transaction processing responsiveness,

(3) Create the necessary framework for efficient control and interface of TPAPs to a common data base,

(4) Provide a specialized, open-ended environment that can be tailored to specific transaction processing needs.

In addition, the operating system allows a reduction in the number of GCOS Program Numbers dedicated to a site's transaction processing requirements.

TRANSACTION PROCESSING OPERATING SYSTEM

The TPAP Operating System (TPOS) is a specialized system designed to control a subset of the total computer environment for the exclusive execution of Transaction Processing Applications Programs (TPAPs). This operating system is by no means a comprehensive computer operating system; however, it must perform many of the executive or control functions required of the latter. The TPAP Operating System executes under GCOS as a slave program. In this context, it resembles the Time-Sharing System.

The TPAP Operating System interfaces with the Transaction Processing System as a TPAP. Consequently, the TPOS is known to the Transaction Processing Executive (TPE) as just another TPAP. This arrangement allows several TPAP Operating Systems to be incorporated into the Transaction Processing System.

Transaction Processing Operating System Executive

The TPAP Operating System Executive consists of those routines that perform the executive or control functions which are typical of any standard operating system. In particular, these functions include the initiation, monitoring and control of system operations, resource allocation and processing support.

In the current state of development, the TPAP Operating System and the TPAP Operating System Executive are synonymous, thus the names can be used interchangeably. Henceforth, the TPAP Operating System is referred to as simply the Executive.

Introduction to the Executive

As previously stated, the Executive is a slave program which operates within the Transaction Processing System as a standard TPAP. The Executive functions as the operating supervisor for some number of TPAPs within a GELBAR environment. That is, the Executive overlays the TPAPs into a portion of its allocated memory and passes control to the TPAPs via the MME GELBAR logic. Because the GELBAR changes the BAR to the requested setting, both the Executive proper and any other TPAPs residing within the Executive's core are protected from memory access by the executing TPAP.

TPAPs that are to execute under the control of the Executive are generically called Keyword Processors. Since the Keyword Processors can be written exactly like a normal TPAP, this label serves solely to differentiate between the two methods of execution.

The Keyword Processors are maintained on high speed storage from which they can be retrieved when requested by an input transaction. Multiple copies of the same Keyword Processor are possible since each input transaction designates to the Executive which Keyword Processor is to be executed, independent of the other transactions being serviced. Thus each transaction effectively gets its own copy of the requested Keyword Processor.

All files must be allocated to the Executive, since the Keyword Processors are loaded without an SSA. This requirement allows all the Keyword Processors to access any data base allocated to the Executive and creates the necessary framework for efficiently controlling multiple data base access requests.

The Executive is responsible for the following major functions:

o Receiving multiple input transactions via Intercom I/O from TPE and scheduling the Keyword Processor identified by the transaction.

o Allocating core and processor time on a priority basis to the scheduled Keyword Processors.

- Processing Keyword Processor program faults and service requests by trapping, identifying and routing the fault to the applicable Executive routine. In particular, all Keyword Processor requests for GCOS system services are trapped, since MMEs and DRLs generate faults. Thus all I/O requests are intercepted and handled by the Executive.

- Collecting and queueing output Intercom messages from the Keyword Processors and initiating Intercom I/O for each message when the processing for its related transaction is complete.

Overview

The Transaction Processing Operating System Executive monitors and controls its computer resources in order to enable a set of programs, the Keyword Processors, to operate concurrently. Its structure can be logically divided into a set of modular routines whose functions are briefly described in the following paragraphs.

Input processing is performed for the Executive by the Transaction Scheduler. The Transaction Scheduler will request a message from the TPE whenever sufficient Input Intercom Buffer space and a free Transaction Attributes and Status Kernel (TASK) are available. A TASK functions as the focal point for all the dynamic information required to control the processing of a transaction by its associated Keyword Processor. A TASK is assigned to each transaction received by the Executive.

When an input message is accepted, the Keywords List is scanned. This list contains the keywords for all Keyword Processors assigned to the Executive. If the keyword is not in the list, an error message is returned to the user and the transaction is deleted. Otherwise, the Keyword Processor attributes are retrieved from the Keyword Processor Profile table and inserted into the TASK. The TASK is then linked to the Core Allocator's Core-Queue and an attempt is made to request another message.

The Core Allocator selects TASKs in priority order and attempts to allocate a sufficient amount of core in which the Keyword Processor can execute. It will generally use the smallest block of available core to make this allocation. If core is not available, but can be obtained by swapping-out an in-core Keyword Processor that meets the criteria for swap-eligibility, then the Keyword Processor swap is initiated. Otherwise, the TASK is bypassed and the next TASK in priority sequence is considered for allocation. This process is repeated until the end of the Core-Queue is reached. The bypass count (initially set from the Keyword Processor Profile) of each TASK in the queue is reduced by one whenever core is allocated to a lower priority TASK. If the bypass count for a particular TASK is zero or is reduced to zero, no lower priority TASK will be considered by the Core Allocator. When no more TASKs can be allocated core, control is given to the Dispatcher.

The Dispatcher is responsible for assuring that all Executive functions are performed. It receives control every time a Keyword Proessor terminates, requests an Executive service, or runs out of time.

The flow of all transactions through the Transaction Processing Operating System is generally accomplished by making an entry in the appropriate queue. For example, the Transaction Scheduler is driven by input messages from TPE and generates an entry in the Core-Queue. The Core Allocator is driven by the Core-Queue and links the TASKs to the Dispatcher's Queue. Message output is done by making entries in the Output Intercom-Queue. A prime function of the Dispatcher is to service the system queues. The Dispatcher effects the appropriate service by enabling the Transaction Scheduler, Core Allocator or other required system routine.

The Dispatcher must also select the highest priority TASK in the Dispatcher's-Queue and give it control via a MME GELBAR, which provides core and time limits to the TASK.

All Keyword Processors are expected to perform I/O via a MME GEINOS request, as is done by most system subroutines such as GFRC. Since GELBAR logic intercepts all MMEs, no Keyword Processor can perform I/O without the Executive's permission. For MME GEINOS, parameters are validated and DCW's are adjusted to account for the GELBAR boundaries. In the case of device I/O, the I/O is reissued by the Executive and the Keyword Processor is roadblocked until the I/O has completed.

For Intercom I/O writes, every Keyword Processor must include an indicator within each output message which specifies if the text is a segment (more text in the message) an End-Of-Message, or an End-Of-Transaction, as dictated by standard TPAP logic. The Executive collects message segments until an End-Of-Message or an End-Of-Transaction status is received. On either status, the Executive links each of the segments successively (no intervening segments from another TASK) into the output stream. It is necessary to link these successively to assure that the segments arrive at their destination in proper sequence.

If the End-Of-Transaction indicator is specified, the Keyword Processor is prohibited from generating additional output messages. However, the Keyword Processor will be given control so it can perform any wrap-up which may be necessary. Following this, standard procedure is for the Keyword Processor to reinitialize itself up to and including its request for another transaction.

At this point the Keyword Processor has completed its processing of the transaction assigned to it; consequently, it is suspended from further execution by the Executive. If the Keyword Processor is reusable (i.e., can process multiple transactions serially), and there is an outstanding input message for the keyword associated with it, the new message is assigned to the Keyword Processor and it is again eligible for execution. Otherwise, reusable or not, the Keyword Processor is terminated and effectively disposed of. In this case and for all non-reusable Keyword Processors, a new copy of the Keyword Processor will be initiated for a subsequent input message that specified the Keyword Processor.

There also exists the capability for the Keyword Processors to communicate among themselves by transmitting an output message with a single Destination-ID of ***. This capability is restricted to internal communication among the Keyword Processors that are controlled by the Executive.

SITE REFERENCE

Introduction

This section is concerned with the necessary site operations that are required to generate, install and run a TPAP Operating System. The major steps that are required to set the system into motion are chronologically listed below:

- o Requesting the user-supplied Keyword Processor programs and characteristics
- o Assembling the Executive with the desired parametric structure and the Keyword Processor characteristics
- o Creating and loading the Keyword Processor Library
- o Installing the Executive within TPE.

The above steps are further explained in the discussions that follow. Those methods presented in the discussions are only meant to be a guideline showing one of many possible methods.

Lastly, it should be emphasized that there can be several TPOSs, since each system externally resembles a normal TPAP.

User Supplied Information

The user is responsible for supplying the Keyword Processor programs and the control parameters, which are required by the Executive. Some of the control information must be coordinated among all users whose transaction processing programs are to be installed under the same Executive.

All Keyword Processor programs should be suitable for a lowload, since the Executive unconditionally lowloads all Keyword Processors. The programs should be submitted in relocatable object form, such as the $ OBJECT deck produced by a compilation.

The user must also supply the control information that defines the Keyword Processor and its file environment. In particular, the user must submit the following:

(1) Keyword-Processor-ID - this is a three character ID that uniquely identifies the Keyword Processor from other Keyword Processors assigned to the Executive,

(2) Maximum input and output message sizes - these sizes should include the message headers and be given in words,

(3) Keywords - these are the keywords to be associated with the Keyword Processor. All Keywords must be eight characters or less in size.

In addition to the above Keyword Processor information, the user must also supply the expected file requirements. This should include the necessary $ FILE control cards along with any $ USERID control cards that would be required to attach the files to the Executive.

Notice that the Keyword-Processor-ID, Keywords and the File-Code assignments should be coordinated among the users to ensure that there will not be any conflicts.

Executive Assembly

The Executive must be assembled to incorporate the necessary Keyword Processor information and to parametrically tailor the Executive to the anticipated transaction processing needs.

All user supplied information regarding the Keyword Processors is inserted into the Executive by means of the Keyword Processor Profile macro. This macro is symbolically denoted by .PRFL.. Actual use of the macro is described in the Keyword Processor Profile discussion.

Of particular importance to the site personnel are the priority and bypass count assignments. Prior to assigning the values, the importance of each Keyword Processor, relative to all other Keyword Processors to be controlled by the Executive, must be established.

Priorities within the Executive are currently pre-emptive, consequently, the priority values are meaningful only in terms of their algebraic relationships and not their individual magnitudes. For example, there would be no operational difference between different sets of Keyword Processor priority assignments as long as the relationships between the assigned values were the same.

Bypass count assignment is more difficult. The purpose of the bypass counts is to allow the Executive to temporarily ignore a priority value during core allocation, thereby allowing more efficient core utilization and system throughput.

The bypass count should be set to zero for only the most urgent Keywords or Keyword Processors. For the remainder of the keywords, the bypass counts should primarily reflect the size of the associated Keyword Processor and secondarily, the amount of time normally required to process a transaction. That is, the larger the Keyword Processor the larger the bypass count; among Keyword Processors of the same size, the slower the processing the larger the bypass count. Of course, a large bypass count should not be assigned to a high priority keyword even if the Keyword Processor is large or slow.

It is anticipated that some experimenting with the bypass counts will be necessary in a heavy transaction processing environment to achieve a satisfactory throughput rate. However, if the processing workload is light to moderate, the effect of the bypass count is negligible.

The parametric structure of the Executive is defined by the series of macros listed below:

.BUFF.	Generate Input/Output Intercom Buffer Space
.COMQ.	Generate Output Intercom-Queue Space
.LINE.	Generate Terminal Control Block & Buffer Space
.MSG.	Set Maximum Input & Output Intercom Message Sizes
.SWAP.	Generate Swap-File Map Space
.TASK.	Set TASK Size & Generate TASK Space
.TPOS.	Set TPOS Options

See Installation Macro usage for an explanation of how to use the above macros.

Keyword Processor Library

A special file must be created for each Executive to hold the Keyword Processor programs which are under its control. This file is called the Keyword Processor Library File and is assigned the file name L-xxx, where xxx is the TPAP-ID assigned to the Executive in TPE. This naming convention allows each Executive to be conveniently associated with its library file.

Creating the Keyword Processor Library

The control card setup to create a library is as follows:

```
1      8      16
______________________________________________

$      SNUMB

$      IDENT

$      FILSYS

USERID TRAX-EXEC$Password

FCREAT TRAX-EXEC/L-xxx,SIZE/n,n/

$      ENDJOB

***EOF
```

where xxx is the TPAP-ID of the controlling Executive.

Generating the Keyword Processor Library

Within the Keyword Processor Library, the program element names for the Keyword Processor programs are of the form S.yyy, where yyy is the Keyword-Processor-ID internal to the Executive. The Executive retrieves the Keyword Processors, during initialization, by performing a GECALL with the desired S.yyy name to the library.

The first Keyword Processor program can be loaded onto the library with the following control card setup:

```
1      8       16
___________________________________________________

$      SNUMB

$      IDENT

$      LOWLOAD

$      OPTION  SAVE/S.yyy,NOGO

$      LANGUAGE

(KEYWORD PROCESSOR PROGRAM)

$      EXECUTE

$      PRMFL   H*,R/W,R,TRAX-EXEC/L-xxx

$      LIMITS

$      ENDJOB

***EOF
```

where yyy is the Keyword-Processor-ID and xxx is the Executive's TPAP-ID.

For subsequent Keyword Processors, the SAVE option must be replaced with the SAVOLD option.

Notice that since TPE requires the TPAP files to be in spawn format, i.e., a control card deck, a Keyword Processor can also be executed as a normal TPAP by inserting a $ PROGRAM card, which specifies the S.yyy Keyword Processor name. Such a deck also has to specify the Keyword Processor Library as a dynamic user's library. In this case a TPAP Profile would also have to exist within TPE for the Keyword Processor.

Executive Installation

Each Executive must be installed into the Transaction Processing System by assembling a TPAP Profile into module TRXD of TPE to specify the operating characteristics of the Executive and by creating and loading a TPAP File to hold the Executive's spawn deck. The installation procedure is described in the Transaction Processing System Site Manual (Order No. DD36).

The necessary operating parameters to be supplied to the TPAP Profile macro are:

(1) Executive's TPAP-ID

(2) Input and output buffer sizes

(3) All keywords belonging to the Keyword Processors assigned to the Executive along with their associated priorities

(4) Accept ***STRT message

(5) BCD input and output

(6) Maintain select/dispatch order

(7) No swap

(8) Accept multiple input messages

The buffer sizes are the sizes used in the Executive's .MSG. macro previously described.

The necessary information that must be included on the TPOSs TPAP File is:

(1) $ LIMITS card specifying the amount of memory to be assigned to the Executive.

(2) $ PRMFL card to attach the Keyword Processor Library as a user's dynamic library with File

Code **.

(3) $ FILE card to attach space for the $L Load-File.

(4) $ FILE card to attach space for the $S Swap-File.

(5) All file cards supplied by the user for the Keyword Processors themselves.

The amount of memory to be allocated to the Executive, as requested on the $ LIMITS card, should take into account the size of the Executive as determined from its assembly.

The $L Load-File holds the Keyword Processors during the Executive's execution, since they can be retrieved from it faster than from the library. Consequently, sufficient space must be allocated to the file to hold all the Keyword Processors. Any excess space is released at the end of the Executive's initialization.

The TPOS TPAP File can contain the Executive in $ OBJECT form or it can specify an object library onto which the Executive has been edited. Independent of the actual method used, a sample control card make-up for the TPAP File is as below:

```
1      8       16
_____________________________________________________

 $     IDENT                               (Optional)

 $     USERID  TRAX-EXEC$Password

 $     LOWLOAD

 $     OBJECT

 (TPOS EXECUTIVE)

 $     DKEND
$    LINK INIT

$    OBJECT

     TPOS INITIALIZATION

$    DKEND

$    EXECUTE DUMP

$    LIMITS  ,Memory-Size

$    PRMFL   **,R,R,TRAX-EXEC/L-xxx  Keyword Processor Library

$    FILE    $L,,Size                Keyword Processor Load-File

$    FILE    $S,,Size                Keyword Processor Swap-File

$    FILE    Keyword
             Processor
             Files

$    FILE

$    ENDJOB
```

Executive Initiation

Frequently used or priority TPAPs are prespawned and remain in the GCOS program queue. Prespawning the priority programs causes them to pass through the relatively slow peripheral and core allocation phase, making them available for immediate execution from the GCOS swap file.

Each Executive can be explicitly spawned by issuing a dummy message with the appropriate keyword or implicitly spawned at the system console or master terminal via the RESTORE *** Command. This command will spawn each TPAP whose profile contains flag word bit 0=1 with the message *** STRT. The Executive recognizes this message, performs its initialization functions and sends an End-of-Transaction status to TPE.

During the spawning phase, those files specified in the TPAP File, are attached to the Executive. These files reflect all files to be managed by the Executive or accessed by the Keyword Processors.

Master Terminal Capabilities

TPOS has master terminal capabilities which are used via a DAC connect to TPOS. There are five master terminal program functions:

- Statistics
- Debug
- Line Switch
- Terminal Hold
- Termination

The master terminal capabilities are entered via a DAC connect. For example, the "JDAC" command in time sharing could be used to connect to TPOS:

```
JDAC TPOS
```

TPOS will issue a connect message in the following format:

```
**TPOS:  MM/DD/YY at HH:MM:SS: ON CHANNEL CCCC

where  MM    = Month
       DD    = Day
       YY    = Year
       HH    = Hour
       MM    = Minute
       SS    = Seconds
       CCCC  = Channel Number
```

At this time TPOS asks which of the four "programs" you wish to enter. The message has the format:

```
PROGRAM NAME ?
```

In response, one may enter:

```
STATS - Statistics

DBUG  - Debug

LSWIT - Line Switch

WAIT - Terminal Hold

BYE - Termination
```

Because of the nature of the information, two of the systems require passwords, which are:

Program	Password
STATS	NUMBERS
DBUG	WELCOME

NOTE: All commands must be entered with "UPPER CASE".

Statistics

TPOS has a master statistics capability. The current statistical functions are:

TPOS	U	List TPOS Usage Information
TPOS	C	List TPOS Communication Counts
TPOS	A	List both TPOS Usage Information and communication Counts
TPOS		Defaults to TPOS A

In addition to the above functions, continuous monitoring may be obtained by the use of an asterisk (*) after the commands. For example:

```
TPOS   U*
TPOS   C*
TPOS   A*
```

Statistical monitoring in continuous mode may be interrupted by use of the break function on a terminal ($*$BRK on a CRT).

Upon completion of the statistical monitoring, the statistical program is terminated by use of the "DONE" command.

Debug

TPOS has a master debug capability. This capability is accessed via a DAC connect to TPOS. The current debug functions are:

Snap

Sa1-a2	Snap from a1 to a2
Sa,n	Snap n words starting at a
SR	Snap processor registers in dump format
SB	Snap breakpoint table

Patch

Pa p1,...,pn	Patch n words starting at a with octal values p1 thru pn
PVa p1,...,pn	Same as above, except verify patch with a snap

Breakpoints

Ba	Insert breakpoint at a
BVa	Insert breakpoint at a and verify
Da	Delete breakpoint at a
DVa	Delete breakpoint at a and verify

Transfer

Ta	Transfer control to a
TVa	Transfer control to a after verification

If at a breakpoint, reload breakpoint processor register values prior to transfer.

Return

R	Return to caller. If at a breakpoint, resume interrupted execution after reloading processor registers.

Caller

C	Display IC where dbug was called. This is useful when called from other than a breakpoint.

Where

W{symbol}	Display location and size associated with the supplied assembly symbol, provided the symbol was assembled into the symbol table.

Offset

O	Display current offset value
Oc	Set address offset to c. The value c will be added to all dbug verb addresses if they are not designated as absolute.

Verification

VON	Set verification mode on
VOFF	Set verification mode off
VASIS	Don't change verification mode

Processor Register Display

X	Display all processor registers in XB,XP, XD order
XB	Display base address registers in XBA,XBB, XBE,XBR order.
SBA	Display MBA contents. If a non-extended memory processor, so indicate.
XBB	Display MBB contents
XBE	Display BER contents
XBR	Display BAR contents
XP	Display 'panel' registers in XA,XQ,XE,XT, XI order
XA	Display AR contents
XQ	Display QR contents
XE	Display Exp-R contents
XT	Display timer contents
XI	Display all index registers in X0 to X7 order
Xn	Display index register n
XD	Display all address registers in XA0 to XA7 order
XAn	Display address register n
XIR	display indicator register

Processor Register Modification

M{V}A p	Modify AR with octal value p and verify if requested
M{V}Q p	Same as above except for QR
M{V}E p	Same as above except for ER
M{V}IR p	Same as above except for IR
M{V}n p	Same as above except for index register n
M{V}An p	Same as above except for address register n

For all modifications, the octal value P cannot be larger than the largest value that can be contained in the register to be modified.

Absolute/Relative Addresses

All dbug addresses are treated as relative to the offset (see O) unless preceeded with the letter A, meaning absolute.

Finished

F	For use with DBUG in a GELBAR environment. This verb effects a DRL to return to the subsystem or program from which it was invoked.

Line Switch

TPOS has a master terminal line switch capability. This function allows the line to be switched from TPOS to another DAC program (such as time sharing). The format is:

```
LSWIT XXXXXX
```

where XXXXXX is the name of the DAC program to which the user wishes to be connected. For example, to switch from TPOS to the Time Sharing System, one would type:

```
LSWIT TSS
```

and the line would switch to the TSS logon sequence.

Terminal Hold

TPOS has a master terminal line hold capability. This function is initiated by typing

```
WAIT
```

and is terminated by using the "break" capability on the terminal ($*$BRK on a CRT).

Termination

TPOS master terminal capabilities may be exited via the "BYE" command which will cause the terminal to be disconnected just as if a logoff had been performed from the Time Sharing System. The format is:

```
BYE
```

Master Terminal Session

The following three pages are representative of the usage of the master terminal functions.

```
SYSTEM ?jdac tpos

** TPOS: 07/25/77 AT 22:49:10 ON CHANNEL 4050

PROGRAM NAME?  stats

[redacted]
READY
?TPOS

TPOS INTERNAL STATUS ON 07/25/77 FROM 21:03:09 TO 22:49:31

PROGRAM-# 15   SNUMB 8912T   ACTIVITY 02   URGENCY 05   SW 000000000000

   TIME           PROCESSOR         MEMORY        I/O           LINES
START 20.762   USED   0.003308   SWAP    7K   USED  0.012   USED     78
ASOF  22.826   LIMIT  0.049623   TOTAL  18K   LIMIT         LIMIT  2048
LAPSE  2.063   GELBAR 0.000222

    .ENDSP - GELBAR DISPATCHES              1538
    .ENGI  - INTERRUPT BROKEN GELBARS         61
    .ENRLQ - RELINQUISHES                   2285
    .ENRLT - RESOURCE LOCKUP THRESHOLDS        0
    .ENTSS - SCHEDULER STALLS                  0
    .ENLRT - LOST INTERCOM READS              87
    .ENLWI - LOST INTERCOM WRITES              0
    .ENRCV - TRANSACTIONS RECEIVED            38
    .ENMLA - MAIN-LEVEL CORE ALLOCATIONS       0
    .ENMLS - MAIN-LEVEL SWAP ALLOCATIONS       0
    .ENSAR - MAIN-LEVEL SWAP REFUSALS          0
    .ENSWP - SWAPS                             0
    .ENNML - NORMAL TERMINATIONS              33
    .ENABT - ABNORMAL TERMINATIONS             5
    .ENIBO - INPUT BUFFER OVERFLOWS            0
    .ENCRI - INPUT BUFFER CORE CHANGES         0
    .ENIOC - COMPLETED INPUT OVERFLOWS         0
    .ENOBO - OUTPUT BUFFER OVERFLOWS           0
    .ENCRO - OUTPUT BUFFER CORE CHANGES        0
    .ENOOC - COMPLETED OUTPUT OVERFLOWS        0




?U@TPOS U

TPOS INTERNAL STATUS ON 07/25/77 FROM 21:03:09 TO 22:50:38

PROGRAM-# 15   SNUMB 8912T   ACTIVITY 02   URGENCY 05   SW 000000000000

   TIME           PROCESSOR         MEMORY        I/O           LINES
START 20.762   USED   0.003376   SWAP    7K   USED  0.012   USED     78
ASOF  22.844   LIMIT  0.049623   TOTAL  18K   LIMIT         LIMIT  2048
LAPSE  2.082   GELBAR 0.000229

?
```

```
WHAT?
?TPOS C

TPOS INTERNAL STATUS ON 07/25/77 FROM 21:03:09 TO 22:51:17

    .ENDSP - GELBAR DISPATCHES               1574
    .ENGI  - INTERRUPT BROKEN GELBARS          61
    .ENRLQ - RELINQUISHES                    2329
    .ENRLT - RESOURCE LOCKUP THRESHOLDS         0
    .ENTSS - SCHEDULER STALLS                   0
    .ENLRT - LOST INTERCOM READS               89
    .ENLWI - LOST INTERCOM WRITES               0
    .ENRCV - TRANSACTIONS RECEIVED             38
    .ENMLA - MAIN-LEVEL CORE ALLOCATIONS        0
    .ENMLS - MAIN-LEVEL SWAP ALLOCATIONS        0
    .ENSAR - MAIN-LEVEL SWAP REFUSALS           0
    .ENSWP - SWAPS                              0
    .ENNML - NORMAL TERMINATIONS               33
    .ENABT - ABNORMAL TERMINATIONS              5
    .ENIBO - INPUT BUFFER OVERFLOWS             0
    .ENCRI - INPUT BUFFER CORE CHANGES          0
    .ENIOC - COMPLETED INPUT OVERFLOWS          0
    .ENOBO - OUTPUT BUFFER OVERFLOWS            0
    .ENCRO - OUTPUT BUFFER CORE CHANGES         0
    .ENOOC - COMPLETED OUTPUT OVERFLOWS         0




?
```

```
WHAT?
?TPOS A


TPOS INTERNAL STATUS ON 07/25/77 FROM 21:03:09 TO 22:52:34


PROGRAM-# 15   SNUMB 8912T   ACTIVITY 02   URGENCY 05   SW 000000000000

    TIME          PROCESSOR         MEMORY        I/O          LINES
START 20.762   USED    0.003500   SWAP    7K   USED  0.014   USED      78
ASOF  22.877   LIMIT   0.049623   TOTAL  18K   LIMIT         LIMIT  2048
LAPSE  2.115   GELBAR 0.000236

    .ENDSP - GELBAR DISPATCHES                 1608
    .ENGI  - INTERRUPT BROKEN GELBARS            62
    .ENRLQ - RELINQUISHES                      2373
    .ENRLT - RESOURCE LOCKUP THRESHOLDS           0
    .ENTSS - SCHEDULER STALLS                     0
    .ENLRT - LOST INTERCOM READS                 90
    .ENLWI - LOST INTERCOM WRITES                 0
    .ENRCV - TRANSACTIONS RECEIVED               38
    .ENMLA - MAIN-LEVEL CORE ALLOCATIONS          0
    .ENMLS - MAIN-LEVEL SWAP ALLOCATIONS          0
    .ENSAR - MAIN-LEVEL SWAP REFUSALS             0
    .ENSWP - SWAPS                                0
    .ENNML - NORMAL TERMINATIONS                 33
    .ENABT - ABNORMAL TERMINATIONS                5
    .ENIBO - INPUT BUFFER OVERFLOWS               0
    .ENCRI - INPUT BUFFER CORE CHANGES            0
    .ENIOC - COMPLETED INPUT OVERFLOWS            0
    .ENOBO - OUTPUT BUFFER OVERFLOWS              0
    .ENCRO - OUTPUT BUFFER CORE CHANGES           0
    .ENOOC - COMPLETED OUTPUT OVERFLOWS           0




?DONE

PROGRAM NAME?  DBUG

[redacted]

<<<ENTER DBUG
DBUG?S)
ERR - ADDRESS NOT OCTAL
DBUG?S0
000000  000000000000
DBUG?F
<<<EXIT DBUG


PROGRAM NAME?  WAIT

START WAITING @ 22:54:52

PROGRAM NAME?  LSWIT TSS
RADC R&D TSS GCOS-GU3 07/25/77 AT 22.919  CHANNEL 4050
```

INSTALLATION MACROS

Several macros are used within TPOS to tailor it to a particular site's operating requirements. Use of the macro along with a description of the macro parameters follows.

.TPOS. - Set TPOS Options

This macro is used to set assembly flags that control the conditional assembly of TPOS modules and tailor TPOS to various operating environments. The macro is used as follows:

```
1      8     16
       .TPOS.  [OPTIONS-STRING-1,...1OPTION-STRING-N]
```

Where option strings currently recognized are:

'SYMBOL-TABLE'	Sets an assembly flag so macro .SYMT. will generate a symbol table.
'CONSOLE=TTY'	Sets an assembly flag indicating that a TTY is to be used in lieu of a system console.
'SYMDEF-SYMBOL-TABLE-ENTRIES'	Sets an assembly flag indicating all symbols encountered by macro .SYMT. should be SYMDEF'd.
'REMOTE-I/O'	Sets and assembly flag to allow conditional assembly of Remote I/O supervisor code.

The .TPOS. macro call should be inserted immediately after the macro definition. See the assembly listing.

Intercom Buffer Space Macro

```
1       8      16
        .BUFF. Input-Intercom-Buffer-Size,
        ETC    Output-Intercom-Buffer-Size
```

This macro establishes the initial sizes of the Intercom input and output buffers in words. As a starting point, the input buffer can be set to the average message size times the number of TASKs desired. Similarly, the output buffer size can be set to the average output buffer size times the number of TASKs. If memory is not a problem, the maximum memory size should be substituted for the average message size in both cases.

This macro should be inserted into the TPOS assembly deck as specified in the program listing.

<u>Output Intercom Queue Macro</u>

```
1      8      16
------------------------------------------------------------
       .COMQ. Number-of-Queue-Entries
```

The single subfield specifies the number of Output Intercom Queue entries in order to reserve space for the complete queue at assembly time. As a guideline, the number of queue entries should be at least as large as the number of TASKs.

This macro should be inserted into the TPOS assembly deck as specified in the program listing.

<u>Terminal Control Block & Buffer Space Macro</u>

```
1      8      16
------------------------------------------------------------
       .LINE. Number-of-Terminal-Lines,
       ETC    Wait-for-Reconnect-Time
```

The first subfield specifies the maximum number of terminal lines that TPOS is to handle.

The second subfield represents the time interval from terminal disconnect during which all control information is to be held in limbo. This time interval is specified in seconds. The purpose of this field is to allow a reconnect feature in a future enhancement.

This macro should be inserted into the TPOS assembly deck as specified in the program listing.

Set Maximum Input & Output Message Sizes Macro

```
1      8       16
______________________________________________________

       .MSG.   Maximum-Input-Message-Size,
       ETC     Maximum-Output-Message-Size
```

Both subfields specify the message sizes in words. The maximum input and output message sizes should be chosen as the maximum sizes from the user-supplied Keyword Processor specifications.

This macro should be inserted into the TPOS assembly deck as specified in the program listing.

Swap-File Map Space Macro

```
1      8       16
______________________________________________________

       .SWAP.  Number-of-Swap-Map-Entries
```

The single subfield specifies the size of the Swap-Map in terms of the number of map entries desired. No more than 255 entries should be requested since an in-core TASK remains assigned to a program even when it is swapped and the maximum number of TASKs is 255.

The .SWAP. macro can be omitted if a Swap-File will not be allocated to TPOS. This macro should be inserted into the TPOS assembly deck as specified in the program listing.

TASK Space Macro

```
1      8       16
______________________________________________________

       .TASK.  Number-of-TASKs,TASK-Stack-Size
```

The first subfield specifies the maximum number of TASKs which are to be assembled into TPOS. No more than 255 TASKs can be requested.

The second subfield sets the size of the TASK IC&I stack. This subfield can be left null for the current TPOS version.

This macro should be inserted into the TPOS assembly deck as specified in the program listing.

SYMBOL CONVENTIONS

Certain symbol conventions are followed within the Executive both to standardize the symbols by their generic usage and/or content and to avoid confusion. The conventions are currently:

..xxxx	Explicitly Named Location Counter Symbol or Location Counter Origin
.xxxx.	Macro Symbol
.Axxxx	Assembly Parameter or Symbol
.AMRKn	MARK Symbol for Conditional Assembly
.AMSWn	Switch Symbol for Macro Expansions
.BITnn	General Use Bit Symbol
.BTxxx	TASK or TCB Bit Flag
.Dxxxx	DRL Processor Primary Entry Point
.Exxxx	Executive Communication Region Cell
.ENxxx	Executive Accumulated Count
.Gxxxx	MME Processor Primary Entry Point (for standard GExxxx MME symbol)
.Kxxxx	Keyword Processor Prefix Area Cell
.KEYLn	Keywords List Offset Symbol
.PRFLn	Keyword Processor Profile Offset Symbol
.Txxxx	TASK Cell
.TCxxx	Terminal Control Block Offset Symbol
B.Txxx	TASK Bit Flag
B.Axxx	Keyword Processor Attribute Bit Flag

SYMBOL CONVENTIONS

DRLnnx or Dnnxxx	Internal DRL Processor Symbol (for DRL symbol value nn)
MMEnnx or Mnnxxx	Internal MME Processor Symbol (for MME symbol value nn)
YYY.nn	Symbol for Entry Point nn of Module YYY
YYYnnn	Internal Executive Symbol for Module YYY, Symbol Sequence Number nnn

Explicitly Named Assembly Location Counters

Named location counters are used during assembly to order or position coding elements and data, but more importantly to allow macro definition of non-contiguous elements. Current location counter names and element description are:

..EXCR	Executive Communication Region
..XLIT	EIS Transliteration Tables
..EXEC	Executive Proper
..MSG	Executive Error Messages
..PRFL	Keyword Processor Profiles
..KEYL	Keywords List
..TASK	TASK Space
..LINE	Terminal Control Block & Buffer Space
..SYMT	Symbol Table
..BUFF	Input-Output Buffer sizes, Output Intercom Queue, Swap-File Map

Location counter defined assembly elements are sequenced according to the first usage order of the counters. Notice that the origin of each explicit location counter must be properly set when the counter is first defined, i.e., invoked.

Executive Module Symbols

Internal Executive symbols for its modules and their component routines assume the form YYYnnn, where YYY is the module name and nnn is the module symbol sequence number. Current module YYY names are:

AIO	Core Allocator I/O

SYMBOL CONVENTIONS

CAL	Core Allocator
COM	Communication Region
DBG	Interactive Dbug
DRL	DRL Validation/Handler
DSP	Dispatcher
ETX	Edit Transaction Number
EXM	Executive Message Intercom
FLT	Fault Handler
HKP	Housekeeping
IIO	Intercom I/O Handler
KEY	Message Keyword Processing
MAC	Macros
MAP	Map Mechanics
MME	MME Validation/Handler
RIO	Remote I/O Supervisor
RLS	Remote Line Service
SCH	Transaction Scheduler
STA	Interactive Statistics
SYM	Symbols
TRM	Terminator

The following YYY symbols are reserved: CMD, DRL, DVC, LUF, MEM, OFL, ONC, PAR, TAG, TRO and ZOP.

CODING PRACTICES

Register Convention

The only dedicated index register within the Executive's system functions is X7. This is used to hold a pointer to the TASK being processed, if appropriate to the particular function. Though the remaining register usage is not specified, generally the lower numbered registers are treated as the more volatile.

Transfer of Control

Transfer of control among the system functions is currently accomplished via the .CALL., .EXIT. and .GOTO. macros. See transfer of control chapter.

There is no transfer of control register safe-storage conventions within the calling sequence. If safe-storage is necessary it must be done by the calling routine. As an aide to determining which registers are affected, each routine has a preface in the assembly listing that specifies which registers it alters or destroys. Notice that if the called routine references yet another routine, it is necessary to check the preface of the latter routine too.

Register safe-storage has been kept to a minimum wherever possible to reduce this housekeeping overhead. Extra care in modifying or installing a system routine can ensure a minimum of register interference and necessary safe-storage. This applies particularly to those routines which are executed on a frequent basis.

Courtesy-Calls

Normally, courtesy-calls are used as the primary level for processing with the main-level suspended in a roadblocked state. Within the Executive, courtesy-calls are used as a second level for processing in addition to the main-level. Thus the main-level is <u>not</u> roadblocked. The Executive's design dictates that some functions be performed and their component routines be given control at both the main and courtesy-call levels.

<u>Inhibited Code</u>

Inhibited code is necessary within and throughout several system functions. Coding must be so conditioned when a system routine is 'common' to both main and courtesy-call levels and not reentrant or when it references or modifies data that is 'common' to both main and courtesy-call levels.

Non-reentrant system routines are inhibited in order to ensure that they are not busy when called by an interrupting courtesy-call. Determination of whether or not a routine is inhibited must be made by examining the assembly listing.

Data that must be handled with inhibited code is generally a system queue or map. Inhibiting the main-level data reference functions similarly to a programmed gate by eliminating overlaping references and the confusion they could create. Notice that inhibiting is only required at the main-level since a courtesy-call cannot be interrupted by either the main-level or by another courtesy-call.

The need for an inhibited reference in any particular case depends on the type of access intended. For example, inhibiting would not be necessary if the reference is meant to 'just take a look'; however, if the reference is meant to modify or to check for some condition and temporarily preserve it, inhibiting would be required. To aide in determining which items are susceptible to this problem, the item descriptions are keyed according to whether or not courtesy-call reference is made to them.

CODING PRACTICES

TASK Symbol Usage

TASK symbols are relative offset symbols; consequently their difference is absolute. When using these symbols and referencing one TASK cell given a pointer to another (other than the base of the TASK), the pointer offset should be expressed as a symbolic difference rather than a numeric difference. If this is not done and TASK symbols are reshuffled, deleted, etc., the pointer offset will be in error. TASK format flexibility is the primary benefit derived from symbolic definition.

Fault Requested System Services

Keyword Processors can request system services by generating a processor fault with the MME or DRL instructions. Currently only a subset of the standard GCOS MME functions are supported. For a description of the interface between the Executive and this type of system service, see the MME processor section. For a description of the restrictions and conventions to be observed by the service routines themselves, see the Dispatcher's Queue discussion.

TRANSFER OF CONTROL

Transfer of control in the initial version of TPOS was accomplished via TSXn type instructions. As development continued, this method failed to provide the flexibility required by new functions. Two types of IC & I stacks were incorporated to eliminate most transfer of control difficulties.

IC & I Stacks

The first stack is a master or TPOS stack, located at .ESTAK. The stack pointer is located in .EICIS as a tally word. This stack pointer always points to the last entry node in the stack provided the stack isn't empty.

The other type of stack is the TASK stack, located at offset symbol .TSTAK in each TASK. The stack pointer is located at offset symbol .TALLY of the stack. As with TPOS's stack, the stack pointer always points to the last entry mode in the stack.

With both stacks, reference to the last entry mode is accomplished by using the tally stack pointer word as an indirect word.

TRANSFER OF CONTROL

There are three allowable transfers of control mechanism. They are the .CALL., .EXIT., and .GOTO. macros. The latter never involves a stack.

.CALL. Mechanism

The .CALL. macro is used for making an entry in the appropriate IC&I stack, if one is specified, and then transferring control to the desired location. The macro expansion and method for making a stack entry varies according to the stack to be used.

For TPOS's stack, the macro expands into:

```
1      8      16
_____________________________________________

       XED    .EPUSH
       DRL    (transfer location)
```

Where .EPUSH is a fixed instruction pair consisting of:

```
1      8      16
_____________________________________________

       EVEN
.EPUSH STC1   .EICIS,DI
       TTF    1,IC*
```

For a TASK stack ,X7 must point to the applicable TASK. In this case the macro expands into:

```
1      8      16
_____________________________________________

       XED    .ECALL
       DRL    (transfer location)
```

Where .ECALL is a fixed instruction pair consisting of:

```
1      8      16
_____________________________________________

       EVEN
.ECALL STC1   .TPUSH,7*
       TTF    1,IC*
```

and .TPUSH is a NOP within the applicable TASK used for making a TASK stack entry. This call consists of:

```
1      8      16
_____________________________________________

.TPUSH NOP    .TALLY,DI
```

.TPUSH is a necessary part of this sequence since it holds the absolute address of .TALLY,7 (relative to TPOS's LAL) thereby making it's address modification field available for the DI type tally modification. If the TASK is moved, .TPUSH must be adjusted to reflect the new location of .TALLY,7.

Overflow protection exists for either stack. If a stack overflow occurs, the TTF transfer at .EPUSH or .ECALL+1 will not be taken. This causes the DRL in either macro expansion to be executed. There is no confusion between a stack overflow DRL and a service DRL, since the latter is returned through the GELBAR Fault Vector instead of the DRL Fault Vector in TPOS's SPP.

.CALL. USAGE

The .CALL. macro is used as follows:

```
1      8       16
__________________________________________________________

       .CALL   |Symbol, Null                |
               {                            }, Call-Mode-
                                                Designator
               |Module-Name, Entry-Point|
```

Where the Call-Mode-Designator specifies the type of call to be made and consequently affects the inline macro expansion, Call-Mode-Designators are:

Blank or T	Use IC&I stack in TASK pointed to by X7.
E	Use TPOS's IC&I stack.
Xn	Do not use any stack. Instead generate a TSXn to the specified symbol or module-name, entry point.

The last Call-Mode-Designator was included to allow existing TSXn transfer of control calls to be inverted to the .CALL. format. This designator is not intended for future use except when calling a module which does not use a stack to exit.

.EXIT. Mechanism

The .EXIT. macro is used for removing an entry from the appropriate IC&I stack, if any, and then returning control to that IC plus a variable offset. As with the .CALL. macro, the macro expansion varies according to the stack specified in the macro call.

FOR TPOS's stack, the macro expands into:

```
1      8       16
__________________________________________________

       EAX0    (variable-offset)+1
       XED     .EPOP
```

Where .EPOP is a fixed instruction pair consisting of:

```
1       8        16
_____________________________________________________________

        EVEN
.EPOP   ASX0     .EICIS,I
        RET      .EICIS,ID
```

The first expansion instruction sets X0 to the specified offset from the .CALL. plus one to step the calling IC past the DRL in the .CALL. macro expansion. The two instructions at .EPOP add the desired IC offset to the IC within the stack and then return control to the resulting IC as the stack pointer is popped to the previous stack entry.

For a TASK stack, X7 must point to the applicable TASK. In this case, the macro expands into:

```
1       8        16
_____________________________________________________________

        EAX0     (variable-offset)+1
        XED      .EEXIT
```

Where .EEXIT is a fixed instruction pair consisting of:

```
1       8        16
_____________________________________________________________

        EVEN
.EEXIT  ASX0     .TALLY,7*
        RET      .TPOP,7*
```

This mechanism is the equivalent of the TPOS stack exit mechanism except an indirect word at .TPOP,7 in the TASK is required to effect the ID tally modification for popping the stack. The contents of .TPOP are:

```
1       8        16
_____________________________________________________________

.TPOP   NOP      .TALLY,ID
```

.EXIT. Usage

The .EXIT. macro is used as follows:

```
1       8        16
_____________________________________________________________

        .EXIT.   IC-Offset, Call-Mode-Designator,
```

Conditional-Transfer

The IC-offset represents the number of instructions to be shipped past the .CALL. calling sequence. The Call-Mode-Designator tailors the macro expansion as follows:

Null or T	If the third parameter is not null, the two instructions Conditional-Transfer 2,IC TRA 3,IC are generated to allow a conditional exit. These instructions are immediately followed by the expansion which returns, using the IC&I in TASK stack.
E	Return using IC&I in TPOS's stack.
N,Xn,*Xn, AU,AL,QU,QL, *AU,*AL,*QU,*QL, I	If the third subfield is null, the single instruction TRA IC-offset, Register-Modification implied by Call-Mode-Designator is generated. If the third subfield is not null, the single instruction. Conditional-Transfer IC-Offset, Register Modification implied by Call-Designator

As with the .CALL. macro, .EXIT. Call-Mode-Designators other than Null, T or E were included for compatability with existing procedures. These designators should be avoided in new procedures.

TRANSFER OF CONTROL

.GOTO. Usage

The .GOTO. transfer of control macro is used as follows:

```
1      8       16
_______________________________________________________

       .GOTO.  |Symbol, Address-Modification|
               {                            } , Condition-
                                                al-Transfer
               |Module-name, Entry-Point    |
```

The Module-name, Entry-Point option will generate:

```
|         TRA          |   |Symbol, Address-Modification|
{         or           }   {             or             }
|Conditional-Transfer  |   |Module-Name, Entry-Point    |
```

Where allowable address modification types are:
N,Xn,I,AU,AL,QU,QL,*Xn,*AU,*AL,*QU,*QL

Precaution

Both .CALL. and .EXIT. macro expansions can consist of more than one instruction. As a result, care must be taken when calling modules with multiple returns since a .CALL. followed by another .CALL. or a .EXIT. could result in a return to an instruction interior to an expansion.

COMMUNICATION REGION

EXECUTIVE COMMUNICATION REGION

This region of the Executive is used for common storage and for communication among the Executive's internal functions. As such, the region contains the following:

- o assembled constants that define the Executive's parametric structure
- o status of internal Executive functions
- o control information for internal functions.

Symbolic tags for Communication Region cells are of the general form .Exxxx.

Special Usage

Some Communication Region cells are referenced within both main and courtesy-call level processing. These cells must be handled with extra care as dictated by the type of reference being made. In order to identify these cells it is sufficient to indicate which ones are accessed in the courtesy-call routines. This is done by inserting the key (CC-Fef) after the cell name in the following communication cell descriptions.

.EACQS - Allocator's Core-Queue Service (CC-Ref)

0 17	18 25	26 35
PTR TO .TMEM OF LAST TASK	NO-PASS TALLY	MBZ
UNSUCCESSFULLY SERVICED BY	or SELECTION	
MAIN-LEVEL CORE ALLOCATOR	Q-DEPTH	

The upper half of this cell holds the current service position within the Core-Queue. This position can point to the base of Core-Queue when the next queue-entry eligible for selection is the first entry in the queue. The lower half of this cell holds a demand selection Core-Queue depth tally that indicates the number of TASKs remaining to be serviced up to and including the highest priority no-pass, when positive, and indicates the current Core-Queue depth of the service, when negative. This cell is used to interlace courtesy-call and main-level allocation activities.

.EAIXP - Allocator's Interrupted Execution Phase (CC-Ref)

0 17	18 35
NEGATIVE COUNT OF COURTESY-	MAIN-LEVEL EXECUTION
CALL NO-PASS's	PHASE

The upper half of this word holds a negative count of Core-Queue no-pass's that were linked or forced in courtesy-call. The lower half describes the current phase of the main-level allocator as follows:

0 = not enabled

1 = demand (load) allocation

2 = 'swap' resource allocation.

This cell is used by courtesy-call routines to interlace their functions with main-level allocations functions.

.EBAS5 - Base Address Setting

0 - 17	18 - 29	30 - 35
BASE ADDRESS SETTING	NOT USED	X5 MOD

This word is used to resolve an address given an offset in index register 5 (X5) to the base setting in the upper half of this word. The base setting is the LAL of some Keyword Processor. This cell is set by the Dispatcher at every dispatch.

.ECALL - Make Entry in TASK IC&I Stack

0 - 17	18 - 35
STC1	.TPUSH,7*
TTF	1,IC*

.ECMAP - Core-Map (CC-Ref)

0 - 17	18 - 25	26	27 - 35
FWD CORE-MAP PTR (.TMEM)	# FREE 1024 WORD BLOCKS	0	MBZ
MBZ	SWAP-CORE LAL		
MBZ			
BKWD CORE-MAP PTR (.TLAL)	SWAP-CORE UAL		

Four words representing the base Core-Map entries. The first two words are the first Core-Map entry and the last two words are the last map entry. Base entry format is identical to the general Core-Map entry. (See Core/Swap Map discussion). Swap-core is the core area in which Keyword Processors are loaded and executed. The swap-core UAL is defined to be:

(swap-core LAL) + (swap-core size)

in multiples of 1024-word blocks. At assembly time .ECMAP upper points to .ECMAP+2, and .ECMAP+3 upper points to .ECMAP+1.

.ECNSL - CONSOLE ID

0 - 23	24 - 35
MBZ	TERMINAL ID

One word that holds the Terminal Identification when a TTY is used in lieu of a system console.

.ECOMO - Output Intercom Enabled Flag (CC-Ref)

This flag is used to indicate that Output Intercom I/O is enabled and that the courtesy-call Output Intercom routine is executing, when set to a non-zero value. The flag is controlled by the Output Intercom I/O routines.

.ECOMQ - Output Intercom-Queue Space (CC-Ref)

```
0                           17 18                         35
|PTR TO LAST ASSIGNED OUTPUT |PTR TO FIRST FREE OUTPUT|
|INTERCOM-Q ENTRY SPACE+1    |INTERCOM-Q ENTRY or ZERO|
|(.ECQSZ)                    |                        |
|INTERCOM-Q ENTRY SPACE+1    |INTERCOM-Q ENTRY or ZERO|
```

These cells are used to manage the free Output Intercom-Queue entries as an available chain. The words are defined by the Output Intercom-Queue macro (.COMQ.). .ECOMQ upper is assembled to point to the first word past the assigned space and the lower half is assembled as the location of the first assigned queue-entry. ECQSZ lower holds a bit mask used to allow/disallow Intercom-Queue entry requests within the Dispatcher's-Queue Service. To allow requests, the mask is 'ORed' into .EDQSM, while to ignore requests, the 1's-complement of the mask is 'ANDed' with .EDQSM. The mask bit position is assigned symbolically in accordance with the bit position of the 'Need Output Intercom-Queue Entry' TASK Status Flag (B.TNQE).

.ECQP - Output Intercom-Queue (CC-Ref)

```
0                           17 18                             35
|FWD LINKED OUTPUT INTERCOM- |BKWD LINKED OUTPUT INTERCOM-|
|QUEUE ENTRY PTR or ZERO     |QUEUE ENTRY PTR or ZERO     |
```

This cell holds the forward and backward pointers to the head and tail of the Output Intercom-Queue. Entries linked to the queue are waiting or in transmission.

.ECORQ - Core-Queue (CC-Ref)

```
0                          17 18        25 26       35
|FWD CORE-QUEUE ENTRY PTR  |BKWD CORE-QUEUE ENTRY PTR|
|TO .TMEM or ZERO          |TO .TLAL or ZERO         |
|# OF TASKs LINKED TO CORE-QUEUE           |MBZ      |
|                                          |         |
```

Two words, the first of which holds the link pointers to the first and last entries in the Core-Queue and the second contains a count of the number of TASKs linked to the queue.

.ECQSZ - Output Intercom-Queue Entry Size

The upper half of this cell is set during assembly to the Output Intercom-Queue entry size by the .COMQ. This cell must be paired with cell .ECOMQ. (See .ECOMQ).

.EDAMP - Static Core Allocator Dampers (CC-Ref)

```
0                                                  35
|STATIC CORE-MAP DAMPER                             |
|STATIC SWAP DAMPER                                 |
```

The Static Core-Map Damper is a Core-Queue to Core-Map control that, when on, signifies core is full relative to the current demands in the Core-Queue. The Static Swap Damper is a Core-Queue to 'swapable' core-holes control that, when on, signifies core is full relative to the set of core-holes that would be created if swap-eligible Keyword Processors were swapped-out of core. The dampers are on when they assume a non-zero value. (See Core Allocator discussion).

.EDQSM - Dispatcher's-Queue Service Mask

```
0                            17 18                   35
|MBZ                           |DISPATCHER'S-QUEUE    |
|                              |SERVICE BIT MASK      |
```

This cell is used by the Dispatcher's-Queue Service when scanning the status of TASKs linked to the queue in order to determine if a TASK is roadblocked by or requesting some Executive function. Selected bits within the mask are set off by the applicable service routines when their service cannot be provided. The bits are set on when a service request is possible. Of the bits used in the mask, the bit positions are aligned with the like bit positions symbolically assigned to those TASK Status Flags being scanned.

.EDQSP - Dispatcher's-Queue Service Position (CC-Ref)

0 17	18 35
PTR TO .TFLAG OF NEXT TASK TO SERVICE	SERVICE IN PROGRESS FLAG

This cell holds the location of the next TASK to be serviced by the Dispatcher's-Queue Service in the upper half, and a flag that indicates the queue service is enabled, when non-zero, in the lower half.

.EDSPQ - Dispatcher's-Queue (CC-Ref)

0 17	18 35
FWD DSP-Q ENTRY PTR TO .TPRIO or ZERO	BKWD DSP-Q ENTRY PTR TO .TFLAG or ZERO
# OF TASKs LINKED TO DSP-Q	

Two words, the first of which holds the link pointers to the first and last entries in the Dispatcher's-Queue, and the second contains a count of the number of TASKs which are linked to the queue.

.EEXIT. - Remove Entry from TASK IC&I Stack

0 17	18 35
ASXO	.TALLY,7*
RET	.TPOP,7*

Two words that hold the fixed instruction pair used by the .EXIT. macro to add a return offset to the calling IC in the TASK stack and to logically remove the entry from the stack while returning control to the resulting IC. See transfer of control.

.EGTQS - GELBAR Time Quantum Sum

0 — 35
GELBAR TIME QUANTUM SUM FOR ALL GELBAR'd PROCESSOR TIME

This cell is used to accumulate all processor time used by the Keyword Processors themselves.

.EIMAP - Input Intercom Buffer Map (CC-Ref)

0 — 17	18 — 35
FWD INPUT BUFFER MAP PTR	# OF UNATTACHED WORDS AT THE START OF THE BUFFER
LOC OF LAST INPUT BUFFER CELL	RESERVED

Two words, the first of which holds the Input Intercom Buffer base map entry. The second word is used to record the last buffer cell address and is not required by map mechanics. These words are defined by the Generate Buffer Space macro (.BUFF.). See Buffer Map discussion for a further explanation.

.EINCC - In Courtesy-Call Flag

This cell is used by internal routines that execute at both main and courtesy-call levels to indicate that execution is at the courtesy-call level.

.EINIT - Initialization Vector

.EINIT is the primary SYMDEF entry point to the Executive. This cell holds a startup vector to the Executive's Initialization Routine. The transfer is required because the initialization routine is attached to the Executive as a link, consequently the entry SYMDEF must be to the main link.

.EKEYL - Keywords List (CC-Ref)

0 — 17	18 — 25	26 — 35
PTR TO FIRST KEYWORD	# KEYWORDS	MBZ

This cell is built during assembly and is used by those routines that search the Keywords List. The cell is defined during assembly after all profiles and keyword entries have been built.

.ELDSP - Last Dispatch

```
0                           17 18                          35
|LOC OF LAST DISPATCHED TASK|NOT USED                       |
```

This word holds a pointer to the last TASK that was dispatched to. This cell is set by the Dispatcher and used by the Fault Handler.

.EMXSZ - Maximum Message Size

```
0                           17 18                          35
|MAXIMUM OUTPUT MESSAGE SIZE|MAXIMUM INPUT MESSAGE SIZE|
```

One word that holds the maximum output message size and the maximum input message size as declared to TPE in the Executive's Profile. This word is used to screen Keyword Processor Profiles during startup, to allocate input buffer space and to ensure that output messages do not exceed the maximum output message size declared to TPE. This cell is defined during assembly by the .MSG. macro.

.ENPQD - No-pass Q-Depth (CC-Ref)

```
0                           17 18          25 26           35
|PTR TO .TPRIO OF HIGHEST    |NO-PASS      |MBZ             |
|PRIORITY  NO-PASS TASK      |Q-DEPTH      |                |
```

The upper half of this cell holds a pointer to TASK cell .TPRIO of the highest priority no-pass in the Core-Queue and the lower half holds the Core-Queue depth of the no-pass. If a no-pass does not exit, this cell is zero.

.ENUQD - Selection Q-Depth (CC-Ref)

```
0                           17 18          25 26           35
|MBZ                         |SELECTION    |MBZ             |
|                            |Q-DEPTH      |                |
```

This cell holds the Core-Queue depth of the last TASK unsuccessfully serviced by the main-level Core Allocator during demand (load) allocation. The Q-depth can be zero when the next TASK eligible to be selected is the first TASK in the Core-Queue. This cell is used by courtesy-call routines to help determine the allocation eligibility of a newly linked demand and to interlace courtesy-call and main-level functions.

.EOMAP - Output Intercom Buffer Map (CC-Ref)

0 17	18 35
FWD OUTPUT BUFFER MAP PTR	# OF UNATTACHED WORDS AT THE START OF THE BUFFER
LOC OF LAST OUTPUT BUFFER CELL	RESERVED

Two words, the first of which holds the Output Intercom Buffer base map entry. The second word is used to record the last assigned buffer cell address and is not required by map mechanics. These words are defined by the Generate Buffer Space macro (.BUFF.). See Buffer Map discussion for a further explanation.

.EORG - TPOS Origin

This symbol under location counter ..EXCR locates the origin of TPOS.

.EPOP - Remove Entry from TPOS's IC&I Stack

0 17	18 35
ASX0	.EICIS,I
RET	.EICIS,ID

Two words that hold the fixed instruction pair used by the .EXIT. macro to add a return offset to the calling IC in TPOS's TASK Stack and to logically remove the entry from the stack while returning control to the resulting IC. See transfer of control.

.EPRFL - Keyword Processor Profiles

0 17	18 25	26 35
PTR TO FIRST KEYWORD PROCESSOR PROFILE	# OF PROFILES	MBZ

This cell points to the first Keyword Processor Profile in the upper half and holds the number of profiles assembled in the lower half. This number is scaled for use as the tally of a Repeat instruction.

.EPRFX - Keyword Processor Prefix

This symbol is used to symbolically define the size of the Keyword Processor Prefix Area, that is the equated value of the symbol is the prefix size. There is <u>no</u> storage associated with this symbol.

.EPUSH - Make Entry in TPOS IC&I Stack

0 17	18 35
STC1	.EICIS,DI
TTF	1,IC*

Two words that hold the fixed instruction pair used by the .CALL. macro to make an IC&I entry in TPOS's stack. This stack is located at .ESTAK and the stack pointer is a tally word at .EICIS. See transfer of control.

.EQMIN - Core-Queue/Core-Map Fence (CC-Ref)

0 17	18 35
MBZ	CORE-Q :: CORE-MAP FENCE

This cell is an internal Core Allocator control that functions as a fence between the demands in the Core-Queue and the available core-holes in the Core-Map. Whenever the Static Core-Map Damper is on, the following relationship is satisfied:

max Core-Map (available core-hole) $\leq$ .EQMIN < min Core-Queue (resource demands)

See the Core Allocator discussion for a further explanation.

.ERLUF - Resources Lock-Up Flag

This is a one word flag that is on when it holds a non-zero value. The flag is set on by the Dispatcher when:

(1) the Dispatcher's-Queue is not empty

(2) it is unable to select a TASK eligible for dispatch

(3) the Output Intercom Processor is not active

(4) no TASKs are roadblocked by outstanding device I/O.

If the Dispatcher cannot clear the flag after one queue service attempt, it initiates lockup threshold logic and commences aborting TASKs to free-up resources. In the event the lockup remains, the Dispatcher will abort the Executive.

.ESDAT - Startup Date

```
0                                                   35
|STARTUP DATE (MMDDYY)                               |
```

One word that holds the startup date at the completion of initialization.

.ESDQP - Swap Search Dispatcher's-Queue Position (CC-Ref)

```
0                       17 18                       35
|PTR TO .TPRIO OF NEXT TASK|MBZ                      |
|WHERE SWAP-ELIGIBLE TASK  |                         |
|SEARCH IS TO RESUME       |                         |
```

This cell is used during main-level swap resource allocation to hold a pointer to cell .TPRIO of the next TASK where the swap-eligible TASK search is to resume.

.ESMAC - Swap-File Map Space (CC-Ref)

0 — 17	18 — 35
PTR TO LAST ASSIGNED SWAP-FILE MAP ENTRY SPACE+1	PTR TO FIRST FREE SWAP-FILE MAP ENTRY or ZERO
SWAP-FILE MAP ENTRY SIZE	MBZ

Two words used to build or release Swap-File Map entries. The lower half of the first word points to the first available map entry if there is one. These cells are defined by the Generate Swap-File Map space macro (.SWAP.). .ESMAC upper is assembled to point to the first word past the assigned space and .ESMAC lower is assembled to point to the first assigned Swap-File Map entry.

.ESMAP - Swap-File Map (CC-Ref)

0 — 17	18 — 35
FWD SWAP-FILE MAP PTR	# OF FREE 64 WORD BLOCKS
MBZ	SWAP-FILE LAL
MBZ	
BKWD SWAP-FILE MAP PTR	SWAP-FILE UAL

Four words that hold the Swap-File base map entries. The first two words are the first Swap-File Map entry and the last two words are the last entry. Entry format is identical to the general Swap-File Map entry format. The Swap-File UAL is defined to be:

(Swap-File LAL) + (Swap-File size)

in 64-word blocks. At assembly time .ESMAP upper points to

.ESMAP+2, and .ESMAP+3 upper points to .ESMAP+1.

.ESORG - Master IC&I Stack Origin

This symbol locates the beginning of the Master IC&I stack used for transfer of control between TPOS executive routines. See .ESTAK.

.ESTAK - Master IC&I Stack

This symbol locates the first word past the Master IC&I stack used for transfer of control between TPOS executive routines. The size of this stack is established as the difference between .ESTACK and .ESORG. This symbol is used to locate the stack since entries are inserted from high to low address, i.e., via DI-type tally modification.

.ESTAL - Stalled Scheduler Vector (CC-Ref)

```
0                                                         35
|IC & I WHERE TRANSACTION SCHEDULER STALLED-OUT or ZERO|
```

The Transaction Scheduler uses this cell as a flag, everytime it is enabled by the Dispatcher at the main-level, to determine if it has stalled-out and needs to be restarted. When the scheduler is stalled, .ESTAL is treated as a vector that holds the IC & I where the Transaction Scheduler stalled-out and is to be restarted. When this cell is zero, the scheduler is enabled at the courtesy-call level.

.ESTIM - Startup Time

```
0                                                         35
|STARTUP TIME (1/64 MS)                                 |
```

This word holds the startup time at the completion of initialization.

.ESWPE - Swap Eligible TASK Count (CC-Ref)

```
0                                                         35
|# OF SWAP-ELIGIBLE TASKs or KEYWORD PROCESSORs         |
```

This cell holds the count of the number of TASKs whose associated Keyword Processors are eligible to be swapped out of core. This condition is indicated within a TASK when its B.TESW bit flag is on in cell .TFLAG.

.ESYMT - Symbol Table

```
0                             17 18         25 26              35
|                               |# OF TABLE   |                 |
|SYMBOL TABLE PTR               |  ENTRIES    |MBZ              |
```

This word is used to access the symbol table if one has been built. If no symbol table exists, this word is all zeros. See symbol table description.

.ETACK - Dispatcher's TASK Alarm Clock

```
0                                                              35
|TIME WHEN NEAREST TASK ALARM CLOCK IS TO RING (1/64 ms)|
|ALL ONEs IF NO TASK ALARM CLOCKs ARE SET               |
```

This cell is used as a wake-up indicator for those Keyword Processors that have issued a MME GEWAKE. The alarm clock is set to the nearest time when a TASK alarm clock is to ring and its Keyword Processor awakened. If no TASK alarm clocks are set, this clock is turned off by setting it to -1 (all ones). The Dispatcher's-Queue Service checks if the alarm is ringing.

.ETASK - TASK Space

```
0                             17 18                            35
|PTR TO LAST ASSIGNED           |PTR TO FIRST FREE              |
|TASK ENTRY SPACE+1             |TASK ENTRY                     |
|(.ETKSZ)                       |                               |
|TASK ENTRY SIZE                |B.TNST                         |
```

These cells are used to manage the unassigned TASK entries as an available chain. The cells are defined by the Generate TASK Space macro (.TASK.). .ETASK upper is assembled to point to the first word past the assigned space and .ETASK lower is assembled as the location of the first assigned TASK entry. .ETKSZ lower holds a bit mask used to allow/disallow need-spawn-TASK requests within the Dispatcher's-Queu Service. To allow requests, the mask is 'ORed' into .EDQSM, while to ignore requests, the 1's complement of the mask is 'ANDed' with .EDQSM. The mask bit position is assigned in accordance with the bit position of the 'need Spawn TASK' (B.TNST) TASK Status Flag.

.ETBUF - Terminal Buffer Space (CC-Ref)

0 17	18 35
PTR TO LAST ASSIGNED TERMINAL BUFFER SPACE +1	PTR TO FIRST FREE TERMINAL BUFFER
TERMINAL BUFFER SIZE	MBZ

These cells are used to manage the unassigned terminal buffer space as an available chain. The cells are generated by the .LINE. macro. .ETBUF upper is assembled to point to the first word past the assigned terminal buffer space and .ETBUF lower is assembled to point to the first word of the assigned buffer space.

.ETCBP - Terminal Control Block Chain

0 17	18 35
FWD TCB CHAIN PTR or ZERO	BKWD TCB CHAIN PTR or ZERO

This cell holds the base forward and backward pointers to the head and tail of the TCB chain. If the chain is empty, this word is zero.

.ETCBT - Terminal Control Block Space

0 17	18 35
PTR TO LAST ASSIGNED TCB SPACE +1	PTR TO FIRST FREE TCB
TCB SIZE	MBZ

These cells are used to manage the unassigned Terminal Control Block (TCB) space as an available chain. Both cells are defined at assembly time by the .LINE. macro. .ETCBT upper is assembled to point to the first word past the assigned TCB space and .ETCBT lower is assembled to point to the first word of the assigned TCB space.

.ETIME - System Time

0 35
SYSTEM TIME (1/64 ms)

This cell holds the system time when the Dispatcher's TASK Alarm Clock (.ETACK) was last checked to see if it was ringing. The alarm clock is said to be ringing whenever .ETIME > .ETACK. This cell is used to individually wake-up the sleeping Keyword Processors.

.ETIMQ - Time Quantum

```
0                                                  35
|LAST GELBAR TIME QUANTUM (1/64 ms)                 |
```

This word holds the time quantum set for the last Keyword Processor GELBAR in 1/64 ms. The cell is set by the Dispatcher whenever it issues a GELBAR, except when it reissues a GELBAR that was broken by an I/O interrupt.

.ETKSZ - TASK Size

The upper half of this cell is set during assembly to the TASK entry size by the .TASK. macro. It must be paired with cell .ETASK. (See .ETASK.)

.ETREG - TASK Processor Registers

```
0                       17 18                          35
|PTR TO TASK PROCESSOR    |NOT USED                     |
|REGISTER STORAGE AREA    |                             |
```

The upper half of the cell holds a pointer to the processor register safe-storage area of the last Keyword Processor that was dispatched to. This area starts at .KREG in the Keyword Processor's Prefix. The value of the pointer is relative to the Executive's LAL. This cell is set by the Dispatcher and is used during fault and interrupt processing.

Communication Region Accumulated Counts

Several counters are maintained by the Executive for use in evaluating its overall action and efficiency. The counters are separated from the other Communication Region cells since they are not a necessary part of the Executive's operating structure.

The general symbol for an accumulated count is .ENxxx. A complete word is dedicated to each counter with the count always right-justified in the cell. The accumulated counts and their content are:

.ENABT	Total Abnormal Keyword Processor Terminations
.ENCRI	Core Reductions for Input Buffer Overflow
.ENCRO	Core Reductions for Output Buffer Overflow
.ENDSP	Total Dispatches (GELBARs Issued)
.ENGI	Total GELBARs Interrupted
.ENIOC	Input Buffer Overflows Completed
.ENLRT	Total Lost Read Interrupts
.ENLWI	Total Lost Write Interrupts
.ENMLA	Total Main-Level Core Allocations
.ENMLS	Total Main-Level Swap Allocations
.ENNML	Total Normal Keyword Processor Terminations
.ENOOC	Output Buffer Overflows Completed
.ENRCV	Total Transactions Received
.ENRLQ	Total executive Relinquishes
.ENRLT	Total Resource Lockup Thresholds
.ENSAR	Total Swap Allocation Refusals
.ENSWP	Total Keyword Processor Swap-outs

.ENTSS	Total Transaction Scheduler Stall-outs
.ENIBO	Input Buffer Overflows
.ENOBO	Output Buffer Overflows

TRANSACTION ATTRIBUTES & STATUS KERNEL (TASK)

A TASK is built and assigned to each input transaction received by the Executive. The TASK is used to concentrate transaction related information in one central area, thereby allowing easy access and control of the transaction's processing specifics.

The TASK is linked to all high-usage internal queues by assigning the queue-entry within the TASK itself. With this method, the queues appear as threads of forward and backward linked TASKs. The TASK concept aids queue processing by making all transaction dependent control information appear as if it were in each queue-entry.

TASK space is generated within the Executive by means of the .TASK. macro. This space is assembled under the ..TASK location counter as a contiguous area after the Keyword Processor Profiles. This region is managed by Communication Region cell .ETASK with the TASK size recorded in the upper half of .ETKSZ.

TASK Assignments

The TASK is defined symbolically to allow for future modifications and additions. All TASK symbols are of the form .Txxxx.

The TASK layout, with currently assigned offset symbols and contents, is illustrated on the next page. The layout is followed by a brief description of each TASK entry.

TASK FORMAT

```
      Offset
      Notes

            0        8 9         17 18                             35
.TNUMB      | TRANSACTION NUMBER                                    |
.TSID       | SOURCE-ID                  |L(KEYWORD PROFILE)        |
.TIN        | TRANSACTION INPUT TIME (1/64 ms)                      |
.TLAPS      | ELAPSED PROCESSOR TIME (1/64 ms)                      |
.TTIMS      | TIME OF LAST SWAP-OUT or                              |
            | ELAPSED PROCESSOR TIME SINCE LAST LOAD (1/64 ms)      |
.TTIMQ      | TIME REMAINING FROM LAST GELBAR (1/64 ms)             |
.TBAR       | CORE LAL     |CORE SIZE    |L(.KREG) Executive LAL    |
            |IC Keyword Processor LAL    |I                         |
.TPRIO E    | FWD DSP-Q PTR or           |PRIORITY                  |
            | BKWD CORE-Q PTR            |                          |
.TFLAG      | BKWD DSP-Q PTR             |Q-SCANNED STATUS FLAGS    |
            | SERVICE VECTOR             |ATTRIBUTE & STATUS FLAGS  |
.TMEM       | FWD CORE-MAP PTR or        |CORE-HOLE SIZE or         |
            | FWD CORE-Q PTR             |CORE DEMAND SIZE          |
.TLAL       | BKWD CORE-MAP PTR          |CORE PARTITION LAL or     |
            |                            |BYPASS COUNT              |
.TSWAP      |*|SWAP-FILE BLOCK ADDRESS   |KEYWORD PROCESSOR SIZE    |
.TMSG       | LOC OF INPUT MESSAGE or    |INPUT MESSAGE SIZE or     |
            | FIRST OUTPUT MESSAGE       |ACCUMULATED OUTPUT        |
            | SEGMENT                    |MESSAGE SIZE              |
.TMSG2      | LOC OF LAST OUTPUT         |# OF LINKED INTERCOM      |
            | MESSAGE SEGMENT            |Q-ENTRIES                 |
.TIDCW      | PSEUDO INTERCOM DCW                                   |
.TERRM      | ERROR CODE                 |NOT USED                  |
.TSPWN      | FWD SPAWN TASK PTR         |BKWD SPAWN TASK PTR       |
.TWAKE      | WAKE-UP TIME FOR GEWAKE ALARM CLOCK (1/64 ms)         |
.TCCV       | LOC OF COURTESY-CALL       |TSX7 COURTESY-CALL        |
            | ROUTINE                    |VECTOR                    |
.TPAD       | RESERVED FOR CORE ALLOCATOR                           |
.TAREG 8    | EIS ADDRESS REGISTERS SAVED HERE (8 WORDS)            |
.TEPL  8    | EIS POINTERS & LENGTHS REGISTERS SAVED HERE (8 WORDS) |
.TPUSH      | PTR TO .TALLY              |RESERVED       | DI-MOD   |
.TPOP       | PTR TO .TALLY              |RESERVED       | ID-MOD   |
.TALLY      | PTR TO (.STAK42+SIZE)      |-(STACK SIZE)  | MBZ      |
.TSTAK      | TASL IC&I STACK                                       |
            |                                                       |
```

E   Even offset
8   0 Mod 8 offset

Description of TASK Entries

.TALLY - TASK Stack Tally Word

This cell holds the tally word used to control the TASK stack. It is used for entry pushdown, popup and stack reference. The address portion of the tally word is initialized to point to the last word in the stack plus one, since entries are inserted from high to low address. The tally count is initialized to minus the stack size. This cell always points to the current stack entry, provided the stack isn't empty, so that it can be used as an indirect to reference the stack.

.TAREG - EIS Address Registers

This is an eight word area used to hold the eight EIS address registers if configured. This area must have a 0 mod 8 offset. The format for word n of this area is as follows:

```
BITs  0-23 = c(ARn)
Bits 24-35 = 0
```

.TEPL - EIS Pointers & Lengths Registers

This is an eight word area used to hold the eight EIS pointers and length registers. This area must have a 0 mod 8 offset.

.TNUMB - Transaction number

Binary transaction number assigned to the transaction described by the TASK.

.TPOP - Popup for TASK IC&I Stack

This cell is an indirect word that points to the TASK IC&I stack tally word at .TALLY with ID-type tally modification. The cell is used for removing an entry from the TASK IC&I stack. The pointer to .TALLY is an absolute address relative to TPOS's LAL.

.TPUSH	- Pushdown for TASK IC&I Stack This cell is an indirect word that points to the TASK IC&I stack tally word at .TALLY with DI-type tally modification. The cell is used for making an entry into the TASK IC&I stack. The pointer to .TALLY is an absolute address relative to TPOS's LAL.
.TSID	- Source-ID Bits 0-19 contain the Source-ID specified in the transaction header. Bits 18-35 contain a pointer to the first word of the Keyword Processor Profile associated with the transaction designated keyword.
.TSTAK	- TASK IC&I Stack This symbol locates the TASK IC&I stack, which must be the last entry in a TASK due to the manner in which the stack size is established. The size of the stack is set by the .TASK. macro. The stack is used by the transfer of control macros.
.TIN	- Time-In Transaction input time (in 1/64th millisecond pulses) when received by TPE. Time is obtained from the transaction header.
.TLAPS	- Elapsed Processor time Total processor time used by the associated key processor in 1/64 millisecond pulses.
.TTIMS	- Swap Timer Elapsed processor time since last load or swap-in if Keyword Processor is in-core, otherwise time of day of last swap-out. All times in 1/64 millisecond pulses.

.TTIMQ - GELBAR Time Quantum

GELBAR Time remaining from last dispatch, in 1/64th millisecond pulses. Zero if a timer-runout occurred since last dispatch.

.TBAR - Keyword Processor Base Address

Bits 0-8 contain the Keyword Processor's base address (in even multiples of 512 words) relative to the Executive.

Bits 9-17 contain the size (in even multiples of 512 word) of the core partition assigned to the Keyword Processor.

Bits 18-35 contain the pointer to the Keyword Processor's register safe-store area relative to the Executive.

.TICI - Return IC & I

Bits 0-19 contain the instruction counter relative to the Keyword Processor where control is to be returned when its next GELBAR dispatch is paid.

Bits 18-35 contain the Keyword Processor's Indicator Register.

.TPRIO - Priority

Bits 0-19 can contain a backward Core-Queue pointer or a forward Dispatcher's-Queue pointer.

Bits 18-35 contain the transaction's priority. The offset assignment for this cell must always be even.

.TFLAG - Bit Flags

Bits 0-19 contain a backward Dispatcher's-Queue pointer.

Bits 18-35 contain TASK and Keyword Processor Status bit flags. Bit definitions are described later.

.TFLAG+1 - Bit Flag Continuation

Bits 0-19 contain a service vector which holds the IC of a system service or functions.

Bits 18-35 contain Status and Attribute bit flags.

Bit definitions are described later.

.TMEM - Memory

If linked to the Core-Queue:

Bits 0-17 contain a forward Core-Queue pointer,

Bits 18-35 contain the core demand size in 1024-word multiples.

If linked to the Core-Map:

Bits 0-17 contain a forward Core-Map pointer,

Bits 18-35 contain the size of the free hold following the core partition defined by this map entry in 1024-word multiples.

.TLAL - Lower Address Limit

If linked to the Core-Queue:

Bits 0-17 are not used,

Bits 18-35 contain the current bypass count of this demand.

If linked to the Core-Map:

Bits 0-17 contain a backward Core-Map pointer,

Bits 18-35 contain the LAL of the core partition defined by this map entry.

.TSWAP - Swap-File Information

Bit 0 holds a flag to identify on which file the Keyword Processor is saved. The $L load-file is flagged by a zero and the $S Swap-file is flagged by a one.

Bits 1-17 contain the starting Load/Swap file block address, in 64-word blocks.

Bits 18-35 contain the number of blocks needed to hold the Keyword Processor on the $L Load-File.

.TMSG - Input/Out Message Control

For an input message:

Bits 0-17 contain a pointer to the input message within the first word after the applicable buffer map entry.

Bits 18-35 contain the size of the input message in words.

For an output message:

Bits 0-17 contain a pointer to the first output message segment within the Output Intercom Buffer. This points to the first message word, not the buffer map entry.

Bits 18-35 contain the accumulated output message size of all message segments serially linked to the first segment. This size includes each segment's header.

.TMSG2 - Output Message control

Bits 0-17 contain a pointer to the last linked output message segment.

Bits 18-35 contain a count of the number of Output Intercom-Queue entries assigned to this TASK.

.TIDCW - Pseudo Intercom DCW

This word holds the Keyword Processor's

Intercom DCW when input or output Intercom is requested. The DCW's data address is relative to the Executive LAL.

.TERRM - Error Message Code

Bits 0-17 contain the Executive error message code if the TASK or Keyword Processor is abnormally terminated.

Bits 18-35 are reserved.

.TSPWN - TASK Spawn Chain

If zero the TASK does not belong to a Spawn Chain, otherwise:

Bits 0-17 contain a forward Spawn TASK pointer.

Bits 18-35 contain a backward Spawn TASK pointer.

.TWAKE - Wake-up Time

Bits 0-35 contain the time this TASK's Keyword Processor is to be awakened. This cell is set by a GEWAKE.

.TCCV - Courtesy-Call Vector

Bits 0-17 contain the location of the courtesy-call routine.

Bits 18-35 contain a TSX7 instruction to transfer control to the courtesy-call routine and to identify the TASK associated with the completed I/O.

.TPAD - Scratchpad

This symbol is associated with one or more cells that the end of the TASK, depending on the assembled TASK size. The first cell, that is the cell addressed as .TPAD, is reserved for the Core Allocator.

If the TASK is linked to the Core-Queue:

Bits 0-17 contain a pointer to the next lower priority no-pass, or zero if this TASK is the lowest priority no-pass.

Bits 18-35 contain a pointer to the next higher priority no-pass or zero if this TASK is the highest priority no-pass.

If the TASK's Keyword Processor is being swapped-in:

Bits 0-17 are not used.

Bits 18-35 contain the Keyword Processor's required core size.

If the TASK's Keyword Processor is being swapped-out:

Bits 0-17 contain a pointer to the replacement TASK (the TASK to be swapped-in).

Bits 18-35 contain the Keyword Processor's LAL.

TASK STATUS AND ATTRIBUTE FLAGS

TASK Status and Attribute Flags are bit flags. The bit position assignment of the flags is defined symbolically with general symbol formations of B.Txxx for TASK Status Flags and B.Axxx for Attribute Flags.

Only TASK Status Flags are allowed in .TFLAG lower. Flags present in .TFLAG+1 are both Status and Attribute Flags.

Care should be exercised when examining a bit flag(s) to ensure that the proper TASK cell, i.e., .TFLAG+1, is being used.

TASK Status Bit Flags

TASK Status Bit Flags are dynamic flags used to indicate the current state of the TASK and its associated Keyword Processor. By the state of the TASK is meant which queues the TASK is linked to.

TASK Status Flags in .TFLAG are called 'Q-scanned' flags. Their usage is somewhat specialized as explained in the Dispatcher's-Queue and Dispatcher's-Queue Service discussions later.

TASK Status Flags in .TFLAG

B.TABT - In Abort

This flag indicates that the TASK and its associated Keyword Processor (if one exists) are being abnormally terminated by the TASK Terminator.

B.TABW - In Abort Wrap-up

This flag has been defined, but currently is not implemented.

B.TDIO - Device I/O Initiated

This flag is set on when device I/O is requested and subsequently initiated by the Executive. The flag is used in determining Dispatch Select and Swap-out eligibility and, during abort or termination processing, to ensure that all device I/O is completed before core assigned to the Keyword Processor is released.

B.TESW - Eligible for Swap

This flag indicates that an in-core Keyword Processor is eligible to be swapped-out. The flag is required since swapping is not necessarily initiated when the eligibility determination is made. Consequently, the flag is used when searching for an in-core Keyword

Processor that can be swapped-out and replaced.

B.TNAB - Need Abort

This flag indicates that the associated Keyword Processor is awaiting abnormal termination processing.

B.TNBS - Need Output Buffer Space

This flag indicates that the associated Keyword Processor is unable to get its requested amount of Output Intercom Buffer space. This condition suspends the requesting Keyword Processor from further execution until sufficient space can be assigned. The Dispatcher will attempt to restart the service for this TASK during its queue service.

B.TNQE - Need Output Intercom-Queue Entry

This flag is set on when the Output Intercom-Queue is full so that there are no available Output Intercom-Queue entries to be assigned to the associated Keyword Processor. This condition suspends the requesting Keyword Processor from further execution. The Dispatcher will attempt to restart the service for this TASK during its queue service.

B.TNST - Need Spawn TASK

This flag indicates that the associated Keyword Processor is awaiting a skeleton TASK in order to initiate Keyword Processor-to-Keyword Processor Intercom communication. This condition suspends the requesting Keyword Processor from further execution. The Dispatcher will attempt to restart the service for the TASK during its queue service.

B.TSWO - Swap-Out in Progress

This flag is set on when swap-out of the associated Keyword Processor has been initiated and turned off when the swap-out has been completed. The Keyword Processor cannot be dispatched to when this flag is on, even though the TASK remains linked to the Dispatcher's-Queue.

B.TTRM - In Termination

This flag indicates that the TASK and its associated Keyword Processor are being terminated by the TASK Terminator.

B.TWAK - Need Wake-up

This flag is set on when the associated Keyword Processor issues a MME GEWAKE. The wake-up time is kept in TASK cell .TWAKE. The flag is used by the Dispatcher's-Queue Service to awaken the Keyword Processor when its alarm clock rings.

TASK Status Flags in .TFLAG+1

B.TBIO - Building Intercom Output

This flag signals that the Executive has received an output Intercom I/O request from the associated Keyword Processor and that the TASK message pointer word .TMSG describes an output message. The flag is used to help determine whether an output message segment is the first segment received or a subsequent segment which must be linked to a message chain.

B.TFLT - ZOP, CMD, MEM, TAG Fault

This flag is set on whenever a Keyword Processor ZOP, CMD, MEM, or TAG fault occurs. The flag is used in subsequent fault processing to determine whether control should be returned to the Keyword Processor.

B.TINC - In-Core

This flag indicates that the TASK is linked to the Core-Map and its Keyword Processor is in core. As such, TASK cells .TMEM and .TLAL are a Core-Map entry with the upper halves of the cells holding map link pointers.

B.TITY - Input Message Type

This flag indicates that the input message was created as output from another Keyword Processor and not directly by the user. The TASK message pointer in .TMSG no longer points to the Input Buffer but to the (linked) output segments in the Output Buffer. The flag is set on by the Output Intercom I/O routine after a TASK has been built for an output message specifying Keyword Processor-to-Keyword Processor Intercom communication.

B.TLCQ - Linked to Core-Queue

When this flag is on, the TASK is linked to the Core-Queue. As such, TASK cells .TPRIO, .TMEM and .TLAL represent a Core-Queue entry with the upper halves of the first two cells holding queue link pointers.

B.TLDQ - Linked to Dispatcher's-Queue

This flag is set on when the TASK is linked to the Dispatcher's-Queue. As such, TASK cells .TPRIO and .TFLAG represent a queue entry with the upper halves of these cells holding queue link pointers.

B.TNUT - New TASK

This flag is set on when a skeleton TASK is built and assigned to an input message. The flag is used by the set-up swap I/O routine to determine whether a DCW for the Keyword Prefix Area (KPA) is to be built and by the swap-in courtesy-call to determine if KPA initialization is necessary. The flag is set off by the swap-in courtesy-call. The flag is also used during termination processing when the Core-Queue is searched for a new TASK that can be assigned to a transaction reentrant Keyword Processor.

B.TOTY - Output Message Type

This flag is set on when the first Intercom output segment of a message has a single destination of ***. This destination indicates Keyword Processor Intercom communication; therefore, this output is designated as input to another Keyword Processor.

B.TOUC - Output Complete

This flag is set on by the Output Intercom I/O routine when an EOT (End-of-Transaction) status is detected in an output message segment. The flag is used by the Intercom I/O Handler to determine if a Keyword Processor has finished processing.

B.TRLQ - Relinquish/Roadblock Requested

This flag is set on when a Keyword Processor issues a MME GERELC or GEROAD and turned off whenever I/O (device or Intercom) is requested. The flag is used to prevent the Keyword Processor from issuing two consecutive GERELC/GEROADs without an intervening I/O.

B.TWSI - Swap-In in Progress

This flag indicates that the associated Keyword Processor is being loaded or swapped-in. The flag is used during abnormal termination to ensure that all I/O is complete before assigned core is released.

B.TSWD - Swapped-Out

This flag is set on when the associated Keyword Processor has been swapped-out and thus resides on the $S Swap-File. When the flag is on a Swap-Map entry has been assigned to this TASK and TASK cell .TSWAP upper holds the starting block number of the assigned Swap-File space.

Attribute Bit Flags

Attribute Flags are set during the Executive's assembly by the parameters supplied to the Keyword Processor Profile macro .PRFL. The flag settings within each profile are static during execution.

Every time a transaction is received, all Attribute Flags accessible.

Attribute Flags in .TFLAG+1 and Keyword Profile

B.ADNS - Do Not Swap

This flag indicates that the associated Keyword Processor is not to be swapped, but should remain in core until it has completed processing each transaction.

B.ARUS - Reusable

This flag specifies that the Keyword Processor is not disposable, that is, after processing a transaction with normal termination, a new transaction can be passed directly to the Keyword Processor with no external setup or initialization required.

B.ASEM - Suppress Error Messages

This flag causes the transmission of all Executive full text error messages to the user to be eliminated. In their place the Executive issues a caveat to inform the user that the designated transaction's processing has been aborted. This warning also specifies the error code or message number.

B.AOVL - Overlays are part of this Keyword Processor

This flag causes the Executive to search for overlays when a MME GERSTR is executed. During initialization, the Keyword Processor Profile is built via a macro. If overlays are used in a Keyword Processor, the Keyword Processor Profile is expanded to accommodate the necessary overlay information and the B.AOVL Flag is set.

TERMINAL CONTROL BLOCK

Terminal Control Block

A Terminal Control Block (TCB) is allocated and assigned to each terminal that connects to TPOS. All identification, control and status information required to process breaks, I/O and disconnects is grouped into the TCB. In addition, other information necessary to link the TCB with a TASK and provide convenient terminal handling functions is contained in the TCB.

All TCBs are linked together via forward pointers in .TCTID of each TCB and backward pointers in .TCGSS. The forward pointers form a true linked chain, while the backward pointers simply locate the previous TCB to allow linking/unlinking.

TCB space is generated by the .LINE. macro. This space is assembled under the ..LINE location counter together with the terminal buffer space. The TCB space is managed by Communication Region cell .ETCBT. The .LINE. macro is used as follows:

```
1      8        16
-------------------------------------------------------------
       .LINE.   NUMBER-OF-TERMINAL-LINES,
       ETC      WAIT-FOR-TERMINAL-RECONNECT-TIME
```

where the reconnect time is specified in seconds.

TCB Assignment

The TCB is symbolically defined to allow for future modifications and additions. All TCB references must be symbolic. TCB symbols have been assigned the general form of .TCXXX.

The TCB layout, with currently assigned offset symbols and contents, is illustrated on the following page. The Layout is followed by a brief description of each TCB entry.

TCB FORMAT

```
           0                               17 18      23 24     29 30      35
           ______________________________________________________________________
.TCTID     |FWD TCB PTR                       |TERMINAL|TERMINAL ID           |
           |                                  |  TYPE  |                      |
           ______________________________________________________________________
.TCGSS     |BKWD TCB PTR                      |GEROUT SOFTWARE                |
           |                                  |STATUS BIT FLAGS               |
           ______________________________________________________________________
.TCCCV     |RIO CC ROUTINE PTR                |TSX6 INSTRUCTION-              |
           |                                  |NO ADDRESS MODIFICATION        |
           ______________________________________________________________________
.TCSLP     |EXT CC ROUTINE PTR                |TASK PTR                       |
           |                                  |                               |
           ______________________________________________________________________
.TCSTA     |GEROUT STATUS WORD                                                |
           |                                                                  |
           ______________________________________________________________________
.TCDAT     |CONNECT DATE (MMDDYY)                                             |
           |                                                                  |
           ______________________________________________________________________
.TCTIM     |CONNECT TIME (1/64 ms)                                            |
           |                                                                  |
           ______________________________________________________________________
.TCBUF     |BUFFER PTR                        |TALLY                 |MBZ     |
           |                                  |                      |        |
           ______________________________________________________________________
.TCBCW     |BUFFER CONTROL TALLYB WORD                                        |
           |                                                                  |
           ______________________________________________________________________
.TCWHO     |USER                                                              |
           |_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ |
           |IDENTIFICATION                                                    |
           |                                                                  |
           ______________________________________________________________________
```

BKWD	-	Backwards
CC	-	Courtesy-Call
EXT	-	External
FWD	-	Forward
ID	-	Identification
LOC	-	Location
PTR	-	Pointer

Description of TCB Entries

.TCTID - Terminal Identification

Bits 0-17 hold a forward pointer to .TCTID of the next TCB, or zero if this TCB is the only or last TCB.

Bits 18-23 contain the Terminal Type.

Bits 24-35 contain the Terminal Identification.

.TCGSS - GEROUT Software Status

Bits 0-17 hold a backward pointer to .TCTID of the previous TCB, or zero if this is the first TCB.

Bits 18-35 contain status and control bit flags which are described later.

.TCCCV - Courtesy-Call Vector

Bits 0-17 contain the location of the GEROUT courtesy-call routine.

Bits 18-35 contain a TSX6 with no address modification. This instruction is used to identify the applicable TCB when a courtesy-call is paid and to transfer control to the appropriate courtesy-call routine.

.TCSLP - Procedure/Structure Pointers

Bits 0-17 hold a pointer to an external courtesy-call routine which is to be invoked when RIO receives the next courtesy-call for this TCB. If zero, no external courtesy-call is to be paid.

Bits 18-35 hold a TASK pointer to the TASK associated with this TCB, if any.

.TCSTA - GEROUT Status

This word is used to receive all GEROUT I/O status words.

.TCDAT - Connect Date

This word holds the BCD date, in MMDDYY form, when the terminal signed on to TPOS.

.TCTIM - Connect Time

This word holds the time, in 1/64 ms pulses, when the terminal signed on to TPOS.

.TCBUF - Buffer Control Word Refresher

Bits 0-17 point to the 64-word buffer assigned to this TCB.

Bits 18-29 hold a byte tally used to refresh .TCBCW. The tally count is set to 4*64.

Bits 30-35 are zero.

.TCBCW - Buffer Control Word

This word holds a byte tally word used to control data moved to the buffer assigned to this TCB.

.TCWHO - User Identification

Two words reserved for future use to hold the terminal user's identification.

TCB Status & Control Flags in .TCGSS

.BTACK - Acknowledging GEROUT 45 Status

This flag is reserved for future use when acknowledging a status previously returned by the GEROUT generalized remote status inquiry function (operation code 45).

.BTBRK - Outstanding GEROUT Break Status

This flag is used to record a break status on the applicable line. The flag is set in response to a break status returned by the GEROUT generalized remote status inquiry function when there is no outstanding I/O to the terminal. The flag is reset when I/O to the terminal is requested, at which time the break status is returned to the caller without actually attempting any I/O.

.BTCBR - I/O Retry Due to Busy Terminal

This flag is used by RIO routines to allow one retry of terminal I/O that results in a terminal busy status. The flag is set on when retry is in effect, after which it is turned off.

.BTCDW - Waiting for Final Disconnect

This flag is set on when a terminal is disconnected and has passed logon. The flag is used to reserve the TCB until a specified time interval has passed. This implementation anticipates a reconnect function. The flag is examined by Line Service, which initiates the final disconnect and release of a TCB.

.BTCIO - Terminal I/O in Progress

This flag is set on whenever terminal I/O is initiated. The flag is set off within the applicable courtesy-call routine when paid.

.BTCLO - Logon in Progress

This flag is turned on when a terminal connects to TPOS and a TCB is assigned. The flag is set off once the program name question has been successfully written to the terminal. This flag is used in conjunction with disconnect processing. If the flag is on and the terminal is disconnected, the TCB is immediately released. If the flag is off, the TCB is placed in limbo for the preset time interval to allow the user to reconnect. The reconnect feature is not implemented in the current version.

.BTDIS - Outstanding GEROUT Disconnect Status

This flag is used to record a disconnect status on the applicable line. The flag is set in response to the receipt of a disconnect status from the GEROUT generalized remote status inquiry function. Disconnect processing is immediately initiated for this TCB. The flag is used in the event that any I/O is requested for the terminal while the TCB is held in limbo awaiting a possible reconnect. The flag is not reset.

.BTLSW - Line Switch in Progress

This flag is set when a line switch request is made via the program name LSWIT. The flag is used by Line Service to enable the line switch function for status checking purposes. When a status is returned, regardless of the line switch outcome, the flag is reset.

.BTWAI - Terminal in Wait Status

This flag is set on when a wait request is made via the program name WAIT. The flag is tested by Line Service to enable the wait function, since it periodically 'rattles' the

AD-A044 612 HONEYWELL INFORMATION SYSTEMS INC MCLEAN VA FEDERAL --ETC F/G 9/2
TRANSACTION PROCESSING OPERATING SYSTEM (TPOS).(U)
AUG 77 R EWING, J BIELSKI F30602-76-C-0277
UNCLASSIFIED RADC-TR-77-276 NL

2 OF 3
ADA044612

sleeping terminal to reassure the user that his line is still connected. The flag is turned off when a break or disconnect is received on the line.

KEYWORD PROCESSOR PROFILE

Profile Function

Each Keyword Processor Profile contains the necessary information, both user supplied and internally generated, to uniquely identify the Keyword Processor, to locate the program elements on the $L Load-File, to specify the attributes of the Keyword Processor and to declare the keywords that are to be associated with the Keyword Processor. A profile must be defined for each Keyword Processor that is to be installed or attached to the Executive.

Profile Generation

The .PRFL. macro is used to generate the profiles into one contiguous area of the Executive under the ..PRFL location counter and to generate the Keywords list into another contiguous area under the ..KEYL location counter. A .PRFL. macro statement must be supplied for each Keyword Processor that is to be known to the Executive.

Each profile is defined as follows:

```
8       16
_____________________________________________

.PRFL.  Keyword-Processor-ID,

ETC     Maximum-Output-Message-Size,

ETC     Maximum-Input-Message-Size,

ETC     Default-Priority,

ETC     (Keyword-1(Priority(Bypass-Count)),
 .
 .
 .
ETC     Octal-Attribute-Flags,

ETC     (Keyword-Processor-Attribute-1,
      .
      .
      .
      ETC     Keyword-Processor Attribute-J),

      ETC     (Keyword-n(Priority(Bypass-Count))),

      ETC     (Overlay-Name-1,...,Overlay-Name-M)
```

where the profile parameters are further explained below.

The Keyword-Processor-ID is a three character BCI identification, which is unique among all Keyword Processors assigned to the Executive.

The maximum input and output message sizes include all headers that prefix the segments making up the messages. The message sizes are given in words.

The default priority is inserted into an 18-bit field. This value should be assigned in consonance with the default priorities assigned to the other profiles.

The Keyword Processor attributes are designed to describe the type and characteristics of the Keyword Processor and to specify which TPOS options are to be applied to the Keyword Processor. The attributes are specified in the macro call by attribute names, which are:

'Reusable'	=	Reusable Keyword Processor
'Do-Not-Swap'	=	Do Not Swap Keyword Processor
'Suppress-Msg'	=	Suppress Error Messages

Each attribute name has a corresponding B.AXXX attribute bit flag assigned to it. The attribute bit flag values are summed together and this value is inserted into the profile. There is room for nine attributes in the profile, four of which are currently used. Three are explicitly named in the macro call, while the fourth (B.AOVL) is implicitly inserted whenever overlay names are specified. See the TASK description for flag explanations.

Each keyword is supplied with an associated priority and bypass count. If the keyword priority is nulled, the default priority assigned to the profile is used.

The bypass count must be a positive integer or zero. This count has a gross effect on the Core Allocator's handling of all Keyword Processors awaiting core assignment; consequently, the count should be set to zero for only the most urgent keywords. See the Core Allocator discussion.

The Keyword Processor Profile macros must be inserted as a group into the Executive prior to assembly.

Profile Usage

The appropriate profile must be referenced for every legal transaction received by the Executive. At this time some of the profile information is copied to the TASK assigned to the transaction.

The Keyword Processor Profiles are located via Communication Region cell .EPRFL. This cell also contains a count of the number of profiles. Similarly, the Keywords List is located via cell .EKEYL, which also contains the total number of keywords for all profiles.

Keyword Processor Profile as Assembled

Keyword Processor Profiles are assembled into a contiguous block using the dedicated location counter ..PRFL with entry definitions formatted via the .PRFL. macro as follows:

KEYWORD PROCESSOR PROFILE

```
0                          17 18        26 27          35
|PTR TO NEXT PROFILE         |KEYWORD-PROCESSOR-ID       |
|MAXIMUM INPUT MESSAGE SIZE  |MAXIMUM OUTPUT MESSAGE SIZE|
|MBZ                         |PTR TO FIRST KEYWORD       |
|DEFAULT PRIORITY            |FLAGS*       |MBZ          |
```

* Keyword Processor Profile Attribute Flags. See the TASK discussion for an explanation.

In addition, if overlays are used in the Keyword Processor, the following is also added:

```
0    7                   17 18                         35
|OC* |                                                   |
```

* Overlay Count

In addition, the following three words are added for EACH overlay:

```
0                      17 18                         35
|         OVERLAY        |          NAME               |
|                        |                             |
|         MBZ            |          MBZ                |
|         MBZ            |          MBZ                |
```

Keyword Processor Profile at Execution Time

During initialization the Keyword Processors are GECALLed by the Executive using the Keyword-Processor-ID and written out to the $L Load-. The Load-File block address, Keyword Processor size in 64-word blocks and entry address are filled in during this process. If the assembled Keyword Processor Profile is unacceptable or the Keyword Processor cannot be written to the Load-File, the second word of the profile is set to zero.

0 17	18 26	27 29	30 35
NOT USED	KEYWORD-PROCESSOR-ID		
0 | LOAD-FILE BLOCK ADDRESS	KEYWORD PROCESSOR SIZE		MBZ
ENTRY ADDRESS	PTR TO FIRST KEYWORD		
DEFAULT PRIORITY	FLAGS	# NEW TASKs IN CORE-Q	

If the Keyword Processor contains overlays, they are GECALL'd by the Executive using the overlay name(s) and written out to the $ L Load-File. The Load-File block address on the $ L Load-File, overlay entry address, origin and size are written in the Overlay entry. Each overlay entry then contains the following:

0 17	18 35
OVERLAY	NAME
ORIGIN	SIZE (In Words)
ENTRY ADDRESS	STARTING BLOCK *

Keywords List

The Keywords List is generated by the Keyword Processor Profile macro. Keywords for all the Keyword Processors form a contiguous table since the ..KEYL location counter is dedicated to the list and invoked by the .PRFL. macro. The format for an entry in the Keywords List is as follows:

0 17	18 35
KEYWORD (BCD CHARACTERS 0-5)	
KEYWORD (BCD CHARACTERS 6-7)	
PTR TO KEYWORD PROCESSOR PROFILE	USAGE COUNT
KEYWORD PRIORITY	BYPASS COUNT

The usage count represents the number of transactions received that designated the related keyword.

QUEUES

Queues are required by the system functions when their initiation would be untimely and must be delayed or when they are unable to provide or complete their function.

Criteria for queue assignment to individual system functions includes the need for a particular queue-entry format and/or the need for optimized or specialized queue mechanics. As a result some system modules have their own queue, such as the Core Allocator's Core-Queue, while other modules do not, such as the TASK Terminator.

The necessary queues for the latter functions are combined as logical sub-queues into one physical queue. Such is the case with the Dispatcher's-Queue. This queue not only holds TASKs waiting for processor assignment, but also those TASKs requiring termination and selected fault requested system services.

Core-Queue

The Core-Queue links all TASKs awaiting core allocation in a forward and backward linked chain. The TASKs are ordered from highest to lowest priority (.TPRIO lower) in the forward (.TMEM upper) queue pointer direction. Pointers to the head and tail of the queue along with number of TASKs linked to the queue are contained in the base of the queue at cell .ECORQ in the Communication Region.

The Core-Queue entry for all TASKs resides within the TASK itself. The general format for an entry is:

```
        0                 17 18        25 26          35
.TPRIO  |BKWD CORE-QUEUE PTR  |PRIORITY                 |
        |                     |                         |
        ---------------------------------------------------

        ---------------------------------------------------
.TMEM   |FWD CORE-QUEUE PTR   |CORE          |MBZ       |
        |                     |DEMAND        |          |
.TLAL   |NOT USED             |BYPASS COUNT             |
        |                     |                         |
        ---------------------------------------------------
```

where the core demand is the required core size in multiples of 1024-word blocks.

It should be pointed out that a TASK cannot be linked to both the Core-Queue and the Dispatcher's-Queue, since TASK cell .TPRIO is used in both queue entries.

Dispatcher's-Queue

The Dispatchers-Queue links all TASKs waiting for processor assignment, fault requested system services or Executive administrative functions into a forward and backward linked chain. The TASKs are ordered from highest to lowest priority (.TPRIO lower) in the backward (.TFLAG upper) queue pointer direction. Pointers to the head and tail of the queue are contained in the base of the queue at cell .EDSPQ in the Communication Region.

Queue entries are embedded within the TASK they represent. The general format of an entry is:

```
        0                 17 18                        35
.TPRIO  |FWD DSP-QUEUE PTR    |PRIORITY                 |
        |                     |                         |
.TFLAG  |BKWD DSP-QUEUE PTR   |Q-SCANNED STATUS FLAGS   |
        |                     |                         |
        |SERVICE VECTOR       |ATTRIBUTE & STATUS       |
        |                     |FLAGS                    |
        ---------------------------------------------------
```

TASK cell .TPRIO must be defined with an even offset so that cells .TPRIO and .TFLAG form an even word pair. This is done to facilitate the queue processing of the 'Q-scanned TASK Status Flags.

Output Intercom-Queue

All Intercom output messages waiting for transmission are queued in priority sequence to form the Output Intercom-Queue. Each queue-entry describes a complete output message, which can consist of several message segments; the priority assigned to the message and a pointer to the TASK, if any, associated with the message.

The queue is forward linked by message priority and backward linked by originating TASK. Pointers to the head and tail of the queue are maintained in cell .ECQP in the Comunication Region. Management of the available queue-entry space is accomplished via cells .ECOMQ. Queue entry format is:

0 17	18 35
FWD OUTPUT INTERCOM-Q PTR	PRIORITY
PTR TO FIRST MSG SEG	NOT USED
BKWD OUTPUT INTERCOM-Q PTR	PTR TO ORIGINATING TASK or ZERO

CORE & SWAP-FILE MAPS

The Core and Swap-File Maps are dynamic partition maps that record usage by entry-defined partitions rather than static partition maps that indicate fixed partition usage.

The dynamic nature of the maps is enabled by using forward and backward link pointers within each map entry. The link pointers thread the map entries within their assigned spaces. The forward pointers link the entries in order of increasing partition origins, i.e., by core or Swap-File partition lower address limits.

Each entry describes the starting block address of a core or Swap-File partition and the number of free blocks at the end of the partition. The format for the general map entry is:

0 17	18 35
FORWARD MAP POINTER	# OF FREE BLOCKS AT THE END OF THE PARTITION
BACKWARD MAP POINTER	PARTITION ORIGIN

The starting block address for the first free block within a partition is found by subtracting the number of free blocks at the end of the partition from the origin of the next partition, which is contained in the succeeding or forward pointed map entry.

Two special map entries that are <u>not</u> usage-defined, but assembled into the maps are the initial and final map entries. The initial entry is required to describe those blocks at the beginning of core or the Swap-File that do not belong to a usage defined partition. The final entry is necessitated by the mechanics of making and releasing map entries.

The format for the base map entries, which are assembled adjacent to one another, is as follows:

CORE & SWAP-FILE MAPS

```
                 0                     17 18                       35
                 ----------------------------------------------------
             ┌  |FORWARD POINTER TO        |# OF UNATTACHED FREE    |
Initial     {   |FIRST MAP ENTRY           |BLOCKS                  |
Entry       {   |MBZ                       |CORE or SWAP-FILE ORIGIN|
             └  |                          |                        |
                 ----------------------------------------------------
             ┌  |MBZ                                                |
            {   |                                                   |
Final       {   |BACKWARD POINTER TO       |CORE or SWAP-FILE UAL   |
Entry        └  |LAST MAP ENTRY            |                        |
                 ----------------------------------------------------
```

where UAL represents the upper address limit.

The base map entries are located in the Executive's Communication Region with the symbolic tags:

.ECMAP	Base Core-Map Entries,
.ESMAP	Base Swap-File Map Entries.

Core-Map Particulars

Entries for the Core-Map are located within the TASK defining the partition at TASK cells .TMEM and .TLAL. Thus there is no need to explicitly obtain a free entry when making a new map entry.

The block size used in the Core-Map is 1024 words. Block addresses are always positioned within a word as multiples of 1024 words.

The core origin and UAL are determined during initialization using cell 31[10] in the Executive's Slave Program Prefix. At this time the Core-Map base entries are defined, so the origin and UAL are merely inserted into them.

Swap-File Map Particulars

A contiguous block of memory is reserved for the

Swap-File Map entries during assembly by the .SWAP. macro. The boundaries of the assigned area are recorded in Communication Region cell .ESMAC by storing the beginning map address in the lower half of .ESMAC and the last map address+1 in the upper half of .ESMAC.

During execution free entries in the map are chained together to avoid a serial search of the map area for an available entry. A pointer to the first free entry is maintained in .ESMAC lower.

The block size used in the Swap-File Map is 64 words. The first available block on the Swap-File is assumed to be zero.

At startup time GCOS is queried via a MME GEFADD to determine the allocated size of the Swap-File. The number of blocks returned is placed in the lower half of .ESMAP, as the number of unattached blocks at the start of the file, and .ESMAP+3, as the UAL of the file.

INTERCOM BUFFER MAPS

The purpose of the Input and Output Intercom Buffer Maps is to make management of available buffer space possible. To this end, the maps describe buffer usage in terms of partition pointers and available or free hole sizes.

With two exceptions, each entry in the maps specifies the available space that follows a message or message segment in the buffers. Thus the maps record buffer usage by means of message-defined buffer partitions rather than by usage of pre-defined fixed buffer partitions.

There are two permanent map entries that are not usage-defined. These are the base map entry and the last map entry. The primary purpose of the base map entry is to describe unattached space at the beginning of the buffer, while the last map entry is necessitated by map mechanics.

Unlike the other maps employed, all buffer map entries, except the base map entries, reside within the buffers at the beginning of the partition they describe. Therefore, the location of a map entry is also the origin of the partition.

The general format of a buffer map entry is:

0 17	18 35
FORWARD MAP LINK POINTER	# OF FREE WORDS AT THE END OF THE PARTITION
BACKWARD MAP POINTER	MBZ

Both maps are forward linked with backward pointers. Only the forward pointers are linked as a chain. The backward pointers within each entry point to the first word of the previous entry and as such, they do not form a linked chain. The only need for the backward pointers is to facilitate the reassignment of buffer space to the previous entry when the space is released.

The base and last map entries along with the size of the buffers are defined by the .BUFF. macro. This macro also reserves the requested amount of buffer space.

The base map entries are assembled into the Communication Region with the symbolic tags:

.EIMAP Input Intercom Buffer Map,

.EOMAP Output Intercom Buffer Map.

The format of the base map entries is:

```
0                          17 18                          35
 ------------------------------------------------------------
|LOC OF NEXT TO LAST BUFFER|# OF UNATTACHED FREE WORDS|
|CELL                      |AT START OF BUFFER         |
|LOC OF LAST BUFFER CELL   |RESERVED                   |
|                          |                           |
 ------------------------------------------------------------
```

The lower half of the first base entry word is assembled as the total size of the buffer minus 2 (to account for the last map entry, which is always present).

The last map entry is assembled into the last two cells of the buffer with the following format:

```
0                          17 18                          35
 ------------------------------------------------------------
|MBZ                                                     |
|POINTER TO BASE MAP ENTRY |RESERVED                      |
 ------------------------------------------------------------
```

It should be pointed out that the forward map pointers are always greater (address-wise) than the location of the map entries in which they lie, since the map entries reside at the beginning of the partition they describe and the forward chain corresponds to increasing buffer addresses.

The location of the map entry or the assigned buffer space, in which an input transaction or output message lies, is kept in the TASK associated with the transaction or message.

Buffer Map Usage

For the Input Intercom Buffer Map, entries are released and can be built at the main-level; while entries are built and excess space is reassigned at the courtesy-call level.

For the Output Intercom Buffer Map, entries are built at the main-level and released at the courtesy-call level.

For both maps the main-level code that references the maps must be inhibited to avoid link pointer confusion within the maps in the event an interrupt or time-runout occurs. Since both maps share the same build and release map entry routines, it is necessary that the routines be interrupted.

Input Intercom Buffer

The Input Intercom Buffer is used for collection of input transactions received via Intercom I/O from TPE.

Prior to issuing an Intercom read request, the Transaction Scheduler scans the Input Intercom Buffer Map to find the smallest available hole in which the largest expected message, as determined from .EMXSZ, will fit. If a hole is found, a new map entry is built to describe the space and the location of the map entry is recorded in cell .TMSG of the skeleton TASK, which was previously obtained by the scheduler. A DCW is built with a data address pointing to the third word of the available space, i.e., the first word past the map entry, and the Intercom is issued.

Upon completion of the I/O, the space description for the partition into which the input message was read, is adjusted to reflect the actual size of the message as determined from the data address reside in the second Status Return word. The actual message size is also placed in cell .TMSG of the skeleton TASK allocated for this transaction.

After a Keyword Processor has requested the transaction via a trapped MME, the transaction-defined buffer partition is released by combining it with the available space of the previous (lower address) or backward pointed partition and the map entry is unlinked.

Output Intercom Buffer

The Output Intercom Buffer is used for collection and Intercom buffering of output message segments destined for TPE. Since message segments with different transaction numbers cannot be intermixed when sent to TPE, all segments for each transaction must be collected and linked together in the Output Intercom Buffer, until an End-of-Message or End-Of-Transaction status is detected. At this time, an Output Intercom-Queue entry is obtained and built for the message. The Output Intercom Processor will issue successive Intercom writes to pass the message segments to TPE.

Message segments are threaded by transaction number within the Output Intercom Buffer by placing a linkage word in the first word of each message segment. The linkage format is:

```
0                         17 18                        35
_________________________________________________________
|POINTER TO NEXT SEGMENT FOR |SIZE OF THIS SEGMENT IN   |
|THIS MESSAGE or ZERO        |WORDS                     |
_________________________________________________________
```

The beginning and ending addresses of each thread are recorded in TASK cells .TMSG and .TMSG2 of the generating Keyword Processor's TASK.

Buffer Threshold Entries

At initialization, a portion of the available memory for the buffers is set aside to serve as a threshold area. Should the reduced buffers not have enough space for additional messages, TPOS will attempt to grant an additional 1024-word block of memory to the buffer. Since the memory (which must be contiguous to the buffer area) may be in use, the threshold area will be used to serve as an immediate and temporary relief until the full 1024 words can be obtained. See the illustration on the following page.

```
                 ---------------------------------
                 |            INPUT              |
                 |            BUFFER             |
                 .                               .
                 .                               .
                 ---------------------------------
LAST ENTRY       |            MBZ                |
                 |_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|
                 |PTR TO         |               |
                 |.EIMAP         |    RESV       |
INPUT BUFFER     |          THRESH               |
THRESHOLD        |_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|
                 |NUMBER WORDS   | NUMBER OF     |
                 |AT INIT.       | WORDS NOW     |
                 |_ _ _ _ _ _ _ _|_ _ _ _ _ _ _ _|
                 |PTR TO         | UNUSED        |
                 |START BUFFER   |               |
                 ---------------------------------
                 |            INPUT              |
                 |            BUFFER             |
                 |           THRESHOLD           |
                 |                               |
                 ---------------------------------
                 |          AVAILABLE            |
                 |           CORE                |
                 .                               .
                 .                               .
                 |          AVAILABLE            |
                 |            CORE               |
                 ---------------------------------
                 |                               |
                 |           OUTPUT              |
                 .           BUFFER              .
                 .          THRESHOLD            .
                 |                               |
                 ---------------------------------
                 |                               |
                 |           OUTPUT              |
                 |           BUFFER              |
                 .                               .
                 .                               .
                 |                               |
                 |                               |
                 ---------------------------------
LAST ENTRY       |            MBZ                |
                 |_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|
                 |PTR TO          |              |
                 |.EOMAP          |              |
THRESHOLD        |          THRESH               |
ENTRY            |_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|
                 |NUMBER WORDS    | NUMBER OF    |
                 |AT INIT.        | WORDS NOW    |
                 |--------------------------------|
                 |PTR TO          | UNUSED       |
                 |START BUFFER    |              |
                 ---------------------------------
```

KEYWORD PREFIX AREA

The Keyword Prefix Area (KPA) of a Keyword Processor is analogous to the Slave Program Prefix (SPA) of a TPAP. This arrangement allows a TPAP to execute within the Executive as a Keyword Processor with no special restrictions.

The KPA is intended to be the common communication area between the Executive and the Keyword Processor analogous to the SPA. It is also used to hold transaction related information whenever the Keyword Processor is not in execution. For example, the DCWs used for a core-load or swap-out are built in the KPA; also the processor register values are safe-stored in the KPA whenever the Keyword Processor is taken out of execution by the Executive.

No control information can be stored in the prefix when its GELBAR is in effect, since the KPA is within the GELBAR boundaries of its associated Keyword Processor.

SPA Cells Supported as KPA

Certain SPA cells supported by GCOS are not supported by the Executive because the cells may only be necessary for GCOS administrative functions or MME functions which are inappropriate, disallowed or legitimate but not incorporated.

Keyword Processor fault vectors are fully supported. The only other KPA cell that is directly supported is the equivalent of the GELOAD limits, i.e., cell 31[10]. The latter cell is set after a Keyword Processor core-load in case .SETU is not attached and the load limits are used.

The lower load limit is developed as the Keyword Processor size in TASK cell .TSWAP lower plus the KPA size as symbolically equated to .EPRFX. That is the next available lower address. The upper load limit is set to the assigned core-size minus one, where the core-size was saved in .TPAD lower during the core-load. If the lower limit is greater than the upper limit, it is set to the latter.

Executive Module & Routine Descriptions

An overview of the major Executive modules along with the descriptions of each module's component routines are on the following pages. In order to simplify and standarize the routine descriptions, the notation below was used:

EPn	- Entry Point n
RRn	- Routine Return n This type of return passes or returns control back to the calling routine.
RTn	- Routine Transfer n This type of return passes control to a predetermined routine.
CPn	- Calling Parameter n
RPn	- Return Parameter n
L( )	- Location of
Call	- Location or IC from which the call to a routine was made.

GELBAR Fault Handler

When interrupts and faults occur while a GELBAR is in effect, GCOS passes control to the Executive Fault Handler via the GELBAR fault vector located in cell 19[10] of the Executive's SPA. Faults occuring outside of a GELBAR are not processed by this handler.

The major functions of the GELBAR Fault Handler, when given control by GCOS, are to:

(1) Safe-store the processor registers, as set by the interrupted Keyword Processor, into its KPA using Communication Region cell .ETREG,

(2) Safe-store the IC & I and remaining GELBAR time in the interrupted Keyword Processor's .TICI and .TTIMQ TASK cells, respectively

(3) Identify the cause and type of the fault indicated by the GELBAR Accumulated Fault Status in the Executive's SPA,

(4) Call the Dispatcher to reissue the GELBAR if it was broken by an I/O interrupt,

(5) Safe-store EIS address registers and pointers and length registers into TASK areas .TAREG and .TEPL respectively.

(6) Pass control to the applicable fault processing routine if a true fault.

If the GELBAR was broken by a true fault, the elapsed processor time for the affected Keyword Processor is updated in TASK cell .TLAPS and the dispatch time remaining is saved in cell .TTIMQ.

Fault Processing

Processing of all fault types except a MME fault, is done within the GELBAR Fault Handler proper.

GELBAR FAULT HANDLER

For Lockup (LUF), Parity (PAR) and Op Not Complete (ONC) faults, the faulting IC&I is saved in cell 10[10] of the Keyword Processor's KPA and abnormal termination of the Keyword Processor is initiated.

For Divide Check (DVC), Derail (DRL) and Overflow (OFL) faults, the faulting IC&I is stored in the first word of the applicable KPA Fault Vector. The second Fault Vector cell is checked to see if it is zero; if so, the Keyword Processor is aborted. Otherwise TASK cell .TICI is adjusted to point to the second word of the Fault Vector and the Dispatcher is called to reissue the GELBAR to the faulting Keyword Processor.

The first occurrence of Zero Op Code (ZOP), Command (CMD), Memory (MEM) and Fault Tag (TAG) faults is treated identically to the previous group of faults relative to the applicable Keyword Processor Fault Vectors. However, the fact that one of these faults has occurred is stored in the TASK. This is accomplished by setting the B.TFLT bit flag on in TASK cell .TFLAG.

Subsequent ZOP, CMD, MEM or TAG faults will be processed <u>only</u> if the first Fault Vector cell has been zeroed by the Keyword Processor. This is interpreted to mean that the Keyword Processor wants to process these faults itself. Determination of whether this is the first fault occurrence is made by testing the B.TFLT bit flag in TASK cell .TFLAG for an 'on' state. If the cell has not been cleared, the Keyword Processor is aborted.

<u>MME Identification/Validation</u>

MME Identification/Validation is a separate routine that is grouped with the GELBAR Fault Handler. Control passes to this routine when the Accumulated Fault Status indicates that a MME fault type has occurred. This routine first identifies the requested MME service by examining the upper half of the Keyword Processor's MME instruction. If the address is legal and specifies an MME function that has been incorporated into the Executive, control is passed to the appropriate MME processing routine. Otherwise the

Keyword Processor is aborted.

DRL Identification/Validation

DRL Identification/Validation is a separate routine that is grouped with the GELBAR Fault Handler. Control passes to this routine when the Accumulated Fault Status indicates that a DRL fault type has occurred. This routine first identifies the requested DRL service by examining the upper half of the Keyword Processor's DRL instruction. If the address is legal and specifies a DRL function that has been incorporated into the Executive, control is passed to the appropriate DRL processing routine. Otherwise the Keyword Processor is aborted.

GELBAR Fault Handler

FUNCTION:

This routine serves the Executive by identifying the reason for a broken GELBAR and routing control to the appropriate fault processing routine. The elapsed processor time for the affected Keyword Processor is updated if a true fault occurred.

ENTRIES:

EP - GELBAR Fault Handler (FLT000).

Control is passed to this entry point via the GELBAR Fault Vectors (located in word 19[10] of the Executive's SPA) by GCOS.

No calling parameters.

RETURNS:

Control is routed to the applicable fault processing routine according to the fault type within the GELBAR accumulated Fault Status (located in word 25[10] of the Executive's SPA).

GELBAR FAULT HANDLER

GELBAR MME Validation

FUNCTION:

This routine identifies and validates the requested MME parameter within the faulting Keyword Processor. If the parameter is illegal or restricted, the Keyword Processor is marked for abort with the appropriate error code stored in the TASK. If the parameter is legal, control is passed to the requested MME processing routine via a MME vector table.

ENTRIES:

EP1 - GELBAR Non-returnable control passes to this routine exclusively from the GELBAR Fault Handler.

CP1 - L (TASK) associated with the faulting Keyword Processor.

RETURNS:

RT1 - Dispatcher's-Queue Service

If an illegal or restricted MME parameter.

RT2 - Applicable MME Routine.

If a legal MME parameter.

RP1 - L (TASK) requesting MME service.

TRANSACTION

Transaction Scheduler

Function

The Transaction Scheduler is responsible for managing transaction input, validating Keywords and queueing new transaction processing requests for the Core Allocator.

The scheduler and main-body of the Executive execute virtually independent of one another. Consequently, they must interface to communicate new transaction processing requests. The interface is the Core-Queue, with each processing request identified by its assigned TASK.

Introduction

The scheduler is initially enabled by the Dispatcher at the main-level. An Intercom read to TPE is issued at the main-level for the first transaction, after which control is returned to the Dispatcher. From this point on the scheduler's processing is essentially courtesy-call driven. That is, when a transaction is passed to the Executive and the Intercom courtesy-call is paid, a subsequent Intercom read for the next transaction is issued within the courtesy-call to maintain GCOS dispatches directly to the scheduler at the courtesy-call level. In this context, the Transaction Scheduler and the remainder of the Executive run asynchronously.

The scheduler can be forced to disable itself. This occurs when internal conditions make it impossible for the scheduler to issue an Intercom read. In this case it must terminate its courtesy-call without an outstanding Intercom read request and, consequently, the scheduler's courtesy-call processing is also terminated. When this occurs, the Transaction Scheduler is said to have 'stalled-out.

The Dispatcher periodically enables the scheduler at the main-level, regardless of whether the scheduler is functioning normally at the courtesy-call level or is stalled-out. The scheduler uses the enable in an attempt to restart itself in the event it is stalled; otherwise, it ignores the enable by immediately returning control to the Dispatcher.

Stall-Out

The scheduler is forced to explicitly 'remember' where it stalls-out, because it cannot relinquish control. If enabled at the main-level, it must return control to the Dispatcher so that any other remaining Executive processing can be acted on. If enabled within courtesy-call, a relinquish is not allowed. As a result, the scheduler cannot implicitly suspend itself; it must terminate itself.

The scheduler stores its IC&I in Communication Region cell .ESTAL, the stalled Scheduler Vector, everytime it enters a phase of Intercom read initiation which it may not be able to successfully complete. If it stalls-out, .ESTAL will point to the phase where the scheduler should be restarted. If the scheduler successfully passes a problem phase, it retains the accumulated controls in case it stalls-out later. When the scheduler successfully completes all phases, .ESTAL is zeroed and the read can be issued.

When the scheduler is enabled by the Dispatcher at the main-level, it examines .ESTAL to see if it is zero. If so the scheduler returns control and thus ignores the enable. Otherwise, it attempts to restart itself by transferring control to the IC&I previously saved in .ESTAL. Successful or not, the scheduler returns control to the Dispatcher after the restart attempt.

Intercom Read Initation

The Transaction Scheduler must obtain sufficient input buffer space and an unassigned TASK prior to issuing an Intercom read to TPE for the first or subsequent input messages.

The amount of input buffer space required for a new message is always presumed to be the size of the largest possible input message, as recorded in Communication Region cell .EMXSZ. The necessary buffer space is obtained by searching the input buffer space map, based at .EIMAP, for a buffer hole in which the largest possible message can fit. The scheduler stalls if sufficient space cannot be found, otherwise, a buffer map entry is built for the allocated space and the location is stored to prevent its loss in an ensuing stall-out.

An attempt to get an unassigned TASK is made once buffer space has been obtained. The TASK Space management cells at .ETASK are queried to determine if an available TASK exists. If none is available, the schedule stalls out; otherwise, a skeleton TASK is built in the assigned space and the location of the previously obtained input buffer space is saved in TASK cell .TMSG. An Intercom read can now be initiated with a courtesy-call pointer to the Intercom read termination routine.

Intercom Read Termination

When the courtesy-call is paid, the Intercom I/O Status Return is first checked. The Intercom read is reissued if a lost interrupt was returned; otherwise, the new transaction's keyword is extracted and validated. This is accomplished by trying to match the keyword against the Keywords List, which is based in communication cell .EKEYL. If no match exists, the transaction proper is discarded by eliminating all references in the assigned TASK to the buffer space in which the message lies. This is done so that the space can be used for the next Intercom read. The TASK assigned to the erroneous messages is then linked to the Dispatcher's-Queue and flagged for abort. This will cause the appropraite error message to be returned to the user.

If a match is found, the input buffer space assigned to the new message is adjusted to reflect the actual message size as determined from the data address residue in the Status Return. The priority associated with the matching keyword in the Keywords List is assigned to the transaction via its TASK and the Keyword Processor Profile is located. Selected profile information is also copied to the TASK.

The scheduler then calls the Core Allocator to link the new transaction's core demand to the Core-Queue. Depending on the return taken by the link routine, an optional attempt can be made to allocate core to the TASK designated Keyword Processor within the courtesy-call.

At this point the scheduler has finished its processing of the new transaction. Notice that either the assigned TASK has been linked to the Core-Queue or the core-load of the designated Keyword Processor has been initiated. Thus the new transaction processing request has been passed on to the Executive's main-body.

The Intercom read termination then calls the Intercom initiation routine to attempt to start the next input request. Successful or not, control is returned and the courtesy-call is terminated. If the initiation routine stalled-out, it will try to restart itself at the main-level when the Dispatcher enables it.

Transaction Scheduler Entry Points

Symbol	Title
SCH100	Initiate Intercom Read
SCH110	Initiate Input Within Courtesy-Call
SCH120	Initiate Input Within Courtesy-Call With Buffer Space
SCH130	Reissue Intercom Read
SCH200	Input Intercom Courtesy-Call
SCH400	Get & Build Skeleton TASK
SCH500	Get Profile Specifics
SCH510	Get Profile Specifics Given Keyword

Initiate Intercom Read

FUNCTION:

This routine obtains input buffer space for the largest input message the Executive can handle, builds a skeleton TASK, and issues an Intercom read to TPE.

ENTRIES:

EP1 - Initiate Intercom Read (SCH100).

EP2 - Initiate Input within CC (SCH110).

This entry is called by the Input Intercom CC routine to drive the Transaction Scheduler at the courtesy-call-level.

EP3 - Initiate Input within CC with Buffer Space (SCH120).

This entry is called by the Input Intercom CC routine when the previous message was in error and the buffer space can be reused.

CP1 - Pointer to input buffer map entry+2.

EP4 - Reissue Intercom Read (SCH130).

RETURNS:

Return is made to the Call+1. If input buffer space or a free TASK cannot be found, the Transaction Scheduler is marked as being stalled by placing the IC&I in .ESTAL.

<u>Input Intercom CC</u>

FUNCTION:

This routine is the courtesy-call serviced by GCOS at completion of the Transaction Scheduler's input Intercom. Transaction related information is inserted into the skeleton TASK, the keyword is isolated and the Executive's Keyword List is scanned for a match to locate the associated Keyword Profile. If a match is found, Keyword Profile specifics are copied into the TASK, excess buffer space is reassigned, the new TASK is linked to the Core-Q with a possible attempt at core allocation, and Intercom input is initiated to receive the next message queued by TPE. If a match cannot be found, input buffer space is reassigned from the TASK, the TASK is marked for abort with the appropriate error code, and Intercom input is initiated using the reassigned input buffer space.

ENTRIES:

EP1 - Input Intercom CC (SCH200).

This entry is driven by courtesy-call.

RETURNS:

Return is made to GCOS via a MME GEENDC.

Get and Build Skeleton TASK

FUNCTION:

This routine attempts to obtain a free TASK. If one is found, the TASK is cleared to zeros and a Transfer-Set-Index instruction is stored as the Courtesy-Call Vector in cell .TCCV of the skeleton TASK. In addition TASK stack cells .TPUSH, .TPOP and .TALLY are initialized.

ENTRIES:

EP1 - Get and Build Skeleton TASK (SCH400).

No calling parameters.

RETURNS:

RR1 - Call+1, Free TASK not available.

RR2 - Call+2, Skeleton TASK built.

Search Keywords List

FUNCTION:

This routine searches the Executive's Keywords List to find a match with the designated keyword.

ENTRIES:

EP1 - Search Keyword List (SCH410).

CP1 - Keyword to be matched.

RETURNS:

RR1 - Call+1, No-match.

RR2 - Call+2, Keyword match found.

RP1 - Pointer to matching entry in the Keywords List.

Get Profile Specifics

FUNCTION:

This routine searches the Executive's Keyword List given a message keyword. If a match is found, the keyword usage count is updated, the Keyword Profile is located and profile specifics are inserted into the TASK. If no match is found or the requested Keyword Processor could not be properly initialized during startup, the TASK is linked to the Dispatcher's-Q for abort with the appropriate error code inserted in the TASK.

ENTRIES:

Common calling parameters are:

CP1 - L(TASK).

CP2 - (input message).

EP1 - Get Profile Specifics (SCH500).

EP2 - Get Profile Specifics given Keyword (SCH510).

CP3 - Keyword.

RETURNS:

RR1 - Call+1, No keyword match.

RR2 - Call+2, Successful.

CORE ALLOCATOR

Introduction

The Core Allocator is responsible for managing available core and TASK core demands, assigning core to requesting TASK on a priority basis and loading and/or swapping the applicable Keyword Processors into core. The major design objective for the Core Allocator is to maximize the use of assigned core space while preserving the priority structure of the Executive.

Concept & Definition

Allocation means the assignment of a selected available resource to a selected resource demand. The allocation algorithum must specify the demand/resource order and the selection method for both the demand and the available resource.

Core allocation within the Executive can assume one of two basic forms. Depending on whether the demand or resource is selected first, the form is either:

1. a specific (selected) demand is effectively matched against all available core resources, or

2. a specific available core resource is effectively matched against all current demands.

In both forms the matching process is synonymous with a selection method.

The first form of allocation is called 'demand' or 'load' allocation since it results in the simple load of a Keyword Processor. The second form is called 'available resource' allocation. It is also called 'swap resource' or just 'swap' allocation when it results in a Keyword Processor swap-out.

Independent of the type of allocation employed, the sets of current demands and available resources are cataloged into the Core-Queue and Core-Map respectively. Demand/resource selection is effectively made from these sets.

Core-Map: The Resource

All TASKs that have been assigned core space are linked together via their .TMEM and .TLAL cells to form the Core-Map. Each entry in the map specifies the lower address limit (LAL) of its associated core partition along with the size of any unused core space at the end or bottom of the partition. Core-Map entries are ordered in ascending sequence by the LAL of the core space they describe when following the pointers in .TMEM (called the forward direction).

The Core-Map is a dynamic partition map since the number of map entries and the sizes of the partitions they define are variable. Because of this dynamic nature, the first and last entries are pre-defined by assembling them into the 'base' of the Core-Map at Communication Region cell .ECMAP. The first base map entry describes the free core-hole (if any) at the start of assigned core. The last base map entry is a dummy.

The unused core spaces (holes) at the end of the dynamic partitions represent available core resources. core-spaces assigned to swap-eligible TASKs are also considered to be available resources. However, these resources are peculiar in that the spaces are <u>not</u> reflected as free holes in the Core-Map for priority reasons. They are called 'swap' resources so that they can be differentiated from the normal free-hole resource. the Dispatcher's-Queue is scanned for swap-eligible TASKs when it is necessary to locate and examine this special resource.

Core-Queue: The Demand

All TASKs awaiting core allocation are linked together via their .TPRIO and .TMEM cells to form the Core-Queue. A queue-entry consists of cells .TPRIO, .TMEM and .TLAL from each linked TASK. Each entry in the queue specifies the priority of the demand (.TPRIO), the required core size for the Keyword Processor designated by the TASK (.TMEM) and an allocation bypass count (.TLAL), which is explained later. The queue entries are linked by demand priority such that the priority sequence is descending when following the thread pointers in .TMEM (called the forward direction) and ascending when following the thread pointers in .TPRIO (called the backward direction).

During certain phases of allocation it is necessary to know the logical Q-index of the Core-Queue entries so that an explicit comparison of their orders can be made. The *logical Q-index of an entry* is its position within the Core-Queue relative to the forward queue pointers. This is an implicit relationship since the chosen queue construction means the queue entries are threaded together by link pointers to form the queue. The logical Q-index is found by counting the number of entries linked from the base of the Core-Queue (.ECORQ) up through the entry in question, in the forward direction. Because of the method used to determine the logical Q-index of an entry, it is alternately called the entry's 'Q-depth'.

Resource Selection

Resource selection is the selection of an available core resource from the Core-Map or a swap resource from the Dispatcher's-Queue. The criteria for resource selection is dependent upon the demand/resource selection order. For demand allocation, resource selection is 'best-fit', i.e., the selected available core resource is the smallest available that satisfies the selected demand. For resource allocation the selection method is circumstantial. This is best exhibited when the available resource is to be created by swapping-out an in-core Keyword Processor and combining the adjacent core holes (if any) with the swap-freed core. In this case the resource is implicitly selected according to the swap-eligibility of some Keyword Processor.

Demand Selection: Bypass/No-pass Logic

Demand Selection is the selection of a particular core demand from the Core-Queue. It is a design objective to make demand selection priority dependent.

Since the Core-Queue observes priorities, i.e., its entries are priority linked to the queue, priority demand selection simply results in the sequential selection of demands, starting with the high priority tail of the queue. Rigid adherence to such a selection method would result in a pre-emptive allocation procedure. This is undesirable because poor core utilization is a likely side effect.

Bypass logic is employed in demand selection to allow the Core allocator to pass over large core demand TASKs within the Core-Queue for which it is unable to find sufficient core, and to assign core to smaller demand, lower priority TASKs.

Though bypass logic is a batching principle in that it attempts to maximize resource utilization and throughput, its necessity in a compromised interactive environment with limited resources is deemed both reasonable and necessary.

Negating bypass logic is no-pass logic. This is required to ensure that the high core demand TASK will eventually be brought into core. Bypass/no-pass decisions depend on the bypass count assigned to a TASK and consequently its Core-Queue entry when it is linked to the queue. This count is presently obtained from the demand associated Keyword Processor Profile.

The bypass count represents the number of times a TASK can be passed over by the Core Allocator before it stops the Allocator from selecting lower priority demands. This count is decremented by one every time core is allocated to a lower priority demand in the Core-Queue until the bypass count has runout. A TASK or Core-Queue entry whose bypass count has runout (is zero) is called a no-pass since it is logically treated as the end of the queue. A no-pass remains until it is allocated its required core.

It is possible for more than one no-pass to be present in the Core-Queue at the same time. For this reason no-pass TASKs are linked together in descending priority sequence via the .TPAD cell within the no-pass TASKs. A pointer to the highest priority no-pass TASK along with its Core-Queue depth is maintained in the Executive Communication Region cell .ENPQD.

Bypass/no-pass logic does not alter the sequential demand selection, it only conditions the priority nature of selection so that it is possible to 'go around' a demand that can't be satisfied, rather than having to stop. It is worth noting that by setting all bypass counts to zero the bypass/no-pass logic is functionally desengaged, thus restoring the Core Allocator to priority pre-emptive demand selection.

Normally when allocation is attempted, the allocator sifts the Core-Queue up through the highest priority no-pass entry; however, when a new demand is linked to the Core-Queue, bypass logic makes it possible to logically shift the queue by examining certain allocation controls which are described later. In this case the queue need not be physically serviced since the controls are sufficient to guarantee that allocation to the preceding old demands would not be possible. A new demand that is selected in this manner is termed 'Q-eligible'.

Demand Resource Assignment

Demand resource assignment consists of core-hole position assignment and core-hole assignment in that order. Core-hole position assignment refers to where in the selected, available core-hole the requested core is to be positioned.

For demand allocation the selected resource is the best fit of all availables. However, even with a best-fit it is likely that the selected resource will be larger than the size of the demand. In this case the demand is positioned at the top of the hole if the Core-Map entry describing the available resource is the first base map entry or lies in a TASK that is <u>not</u> eligible for swap. If the map entry lies in a TASK that <u>is</u> eligible for swap, the demand is positioned at the bottom of the available core-hole. This procedure prevents the free core-hole created by a swap-out from becoming fragmented since any unused space in the available hole continues to be described by the previous map entry. Thus the demand is positioned at the top of the hole if the previous (lower address) core partition is <u>not</u> a 'swapable' resource, and at the bottom of the hole <u>if</u> the previous is 'swapable'.

For swap resource allocation, the swap hole is combined with the free core-holes above and below it and the demand is positioned at the top of the resultant core-hole.

Core-hole assignment is the actual resource to demand assignment. This is accomplished by unlinking the demand from the Core-Queue, building a Core-Map entry to describe the core partition being allocated including any free space below the demand, and linking the new entry (and the TASK in which it lies) into the Core-Map.

<u>Core-Map Anomaly During Swap Resource Allocation</u>

Within the demand resource assignment phase of swap resource allocation, it is desirable to position the selected demand at the top of the swap hole. Because the free hole above (lower address) the swap hole proper could grow while the swap hole is being freed-up, i.e., while the Keyword Processor occupying the hole is being swapped-out, actual resource assignment is not made to the demand until the swap hole is free. If the demand is larger than the size of the swap hole proper, it is unclear how much of the free core-hole (if any) below the swap hole will be needed.

This problem is resolved during the initial allocation (before the swap hole is freed up) by combining the swap hole with the free holes above and below it via the Core-Map entry describing the swap hole and specifying that there is no free hole at the end of this partition. The map anomaly occurs here since the size of the combined core-holes could be greater than demand. This means there might be a free hole of unknown size at the end of the partition.

The map anomaly is rectified when the swap hole has been freed up and true assignment can be made to the demand. At this time the swap hole is combined with the free hole above it (if any) and the Core-Map entry is adjusted to reflect the actual hole size at the end of the partition. Though the free hole at the bottom of the partition is lost during swap-out, this method ensures that the largest possible hole will be created.

CORE ALLOCATOR LOGIC

Introduction

The Core Allocator is a modular structure consisting of allocation routines, such as the demand/resource selection and assignment procedures, and the necessary support routines, such as linking or unlinking a demand to the Core-Queue, update bypass counts, release core, mark core eligible for swap, load Keyword Processors into core, etc. The allocation routines represent different selection criteria that meet specific needs and respond to key events, all in order to minimize response time and allocation overhead. The support routines are effectively independent of the different allocation methods and consequently are considered 'common' routines.

Major Execution Phases

CORE ALLOCATOR

The Core Allocator has been designed to allocate core and thus to execute at main and courtesy-call levels. Main-level allocation is dedicated to servicing all demands in the Core-Queue, up to the highest priority no-pass, in priority order. Courtesy-call allocation is dedicated to an event selected demand or resource such as a newly received demand or a free core-hole created by a swap-out.

In addition to multi-level execution, the Core Allocator allows the selective interruption of main-level allocation by courtesy-call allocation. This is permitted so that the Core Allocator will be as responsive as posible. Of course, demand priorities are still observed.

Phase Interruption

Interruption creates procedural problems and constraints on the allocator's routines and the handling of the Core-Queue, Core-Map and control information. In particular, common routines must be effectively reentrant; queue, map, and control information must be effectively gated; and the courtesy-call allocations must be interlaced with the main-level allocations so their efforts are compatible and well-defined.

The first two problems are resolved by inhibiting the code in the support routines and any main-level code that alters the Core-Queue, Core-Map or control information. Inhibiting is functionally equivalent to programmed gating. Furthermore, gated code can be handled as reentrant code.

The courtesy-call, main-level interlace is accomplished by interlace controls (described later) and by judiciously inhibiting the allocation proper routines. Interrupt breathers are strategically positioned within these routines to allow the interruptions. At these points, the main-level allocator's state is known and well-defined by the interlace controls; furthermore, the interrupt breathers are chosen to be sufficiently close, execution time-wise, so a lockup fault will not occur because of the inhibited code.

Allocation Controls

Allocation controls are classified by usage into primary controls that directly control allocation initiation or govern demand/resource selection criteria, secondary controls that support the allocation process, and interlace controls that coordinate independent allocation phases. Several controls are necessarily multi-type.

Primary Controls

The objectives of primary controls and their associated logic are to eliminate unnecessary allocation passes and to streamline demand/resource selection. These objectives are met via the Core-Queue/Core-Map Fence and the Static Dampers.

Core-Queue/Core-Map Fence

In its simplest form, allocation consists of the selection of a demand or resource followed by a serial scan of available resources or current demands in order to select one, subject to some criteria such as first-fit, best-fit, etc. This procedure would be repeated for each demand or resource. During design of the allocator it was concluded that the time by a straight forward allocation method, as above, would result in an unacceptable overhead burden on the Executive, even with a small number of demands. This conclusion was based primarily on the expected frequency and desired speed of the allocation process.

The Core-Queue/Core-Map Fence was devised to minimize allocation and selection passes. The fence is an essential allocation control that functions as a selection filter for both demands and resources, along with other uses described later.

To give meaning to the fence, the illustration below depicts the sets of demands and available resources plotted on a scale of appropriate resource units.

```
Available Resources)......... (Resource Demands

Low ...... (Resource Units Scale) ...... High
```

Whenever the set of demands meets the set of availables, allocation would be possible. However, considerable overhead can be spent in determing when or if this condition exists. For example in the simple form of allocation given above, this determination can only be made by examining each element in one set against all the elements in the other set. The problem then is to minimize the allocator's overhead by simplifying this feasibility determination. This is accomplished via the Core-Queue/Core-Map Fence.

The Core-Queue/Core-Map Fence is a logical barrier that lies between the set of availables and the set of demands, subject to the constraint that:

```
Available Resources ......................Resource Demands
Low ....... (Resource Units Scale) ....... High
```

where the fence is located at symbol .EQMIN in the Communication Region.

When a demand violates the constraint by crossing over the fence ($\leq$), either a demand allocation attempt is indicated or, if allocation is progress, the demand is selectable for further consideration. Conversely, when an available resource crosses over the fence (>), resource allocation is indicated or this resource is selectable. Either way the fence allows a ready check of allocation feasibility, though by no means does it guarantee its success.

The fence is initialized during the Executive's initialization to the size of assigned, swapable core allocated to the Executive. During execution, the fence is adjusted in three possible ways:

1. When a demand crosses the fence but cannot be allocated core, the fence is set to the demand size minus one resource unit,

2. When an available resource crosses the fence, the fence is set to the available resource size, or

3. During main-level demand allocation, the fence is set to the largest available hole size if there are four or more demands eligible for allocation.

An additional interpretation that should be applied to the fence arises from the above methods of adjustment. During demand allocation, the fence is treated as an upper bound on the largest available core-hole and is used as a demand selection filter. Notice that the first adjustment method supports this view because it tends to refine the upper bound property. Similarly, resource allocation treats the fence as a lower bound on the smallest demand in the Core-Queue. In this case the lower bound property is refined by fence adjustments.

Because demand allocation is the most prevalent type, main-level 'swap' resource allocation uses the fence as a starting bond only. Subsequent adjustments that would normally be applied to the fence are actually applied to a true demand lower bound internal to the resource allocation routine. This is done to preserve the available resource upper bound property that resulted from the demand allocation pass.

In summary, the fence interpretation is biased towards the upper resource bound property, with the relation

$$\max_{\text{Core-Map}} \left\{ \begin{matrix} \text{Available} \\ \text{Core-Hole Sizes} \end{matrix} \right\} \leq \begin{matrix} \text{Core-Queue/Core-Map} \\ \text{Fence} \end{matrix}$$

always being true. However, it is possible for a demand to temporarily cross the fence, since it will only alter the fence after an unsuccessful allocation attempt.

Dampers: General

The Core Allocator can execute in virtually asynchronous allocation phases because it is both event driven and event cued. The latter case occurs by design at the main-level in order to centralize global calls to the allocator and to eliminate untimely allocation passes.

The combination of cued and immediate allocator enables requires the periodic testing for the presence of a cued enable and the gating of some allocation functions until the enable is activated. The latter requirement is necessary to protect the effect of the event on the gross allocation state.

The statis dampers were devised to control and interlace the external driven and cued calls to the allocation processes and to eliminae main-level allocation passes, while the gross demand/resource state remains static, by signaling when a 'favorable' change of state has occurred.

Dampers: Meaning

There are two dampers at .EDAMP, each occupying one cell. These are called the Static Core-Map Damper and the Static Swap Damper in that order. The dampers are two-valued with the damping or 'on' state represented by a non-zero value, which is the IC&I of who last turned the damper on.

The Static Core-Map Damper summarizes the relationship between the set of demands and set of available (but not swapable) core-holes by means of the Core-Queue/Core-Map Fence. When the damper is on, core assigned to the Executive is compact (packed tight). This means no current demand in the Core-Queue up to and including the highest priority no-pass will fit in any current free core-hole described by the Core-Map given the current hole distribution. This is the basic interpretation of the damper. When the damper's state is tested, the question being asked is whether or not core is compact, or whether or not a logical consequence of either compact allocation state is true. The damper is particularly effective when the initiation of demand allocation is in doubt. In this case an allocation attempt would be fruitless if the damper is on since free core is compact.

The events associated with a possible change in the damper's state are the release of core, the linking/unlinking of a demand to/from the Core-Queue, or the completion of demand allocation. Specifically, the damper is turned off whenever a core-hole crosses the Core-Queue/Core-Map Fence, a demand crosses the fence and no attempt is made to allocate core to it, or the highest priority no-pass is unlinked from the Core-Queue; the damper is turned on when main-level demand allocation complete; i.e., has serviced the Core-Queue up to the highest priority no-pass.

This damper is generally altered by Core-Map changes, so a static Core-Map results in a static damper state. Because of this relationship, the damper is called the Static Core-Map Damper.

The Static Swap Damper summarizes the relationship between the set of demands and the set of 'swapable' core-holes, also by means of the Core-Queue/Core-Map Fence. When this damper is on, core assigned to the Executive is compact up to any 'swapable' holes and their adjacent free core-holes. This is the basic interpretation of the swap damper. Even though this damper's meaning and function are similar to the Static Core-Map Damper, the swap damper is required because 'swapable' core resources are not described by the Core-Map.

The swap damper's state is affected by the release of core and the linking/unlinking of a demand to/from the Core-Queue similar to the Core-Map damper. In addition,the state can be altered by designating an assigned area of core as 'swapable' or by 'swap' resource allocation. Specifically, the damper is turned off whenever a 'swapable' core-hole including adjacent free holes crosses the Core-Queue/Core-Map Fence, demand allocation to a newly linked demand is not possible or unsuccessful and swap resource allocation is not attempted, or the highest priority no-pass is unlinked from the Core-Queue; the damper is turned on when main-level resource allocation is initiated.

For reasons analogous to the Static Core-Map Damper's relationship to the Core-Map, the Static Swap Damper generally mirrors any periods during which 'swapable' core resources remain static.

Both dampers are set on if there are no demands present, that is the Core queue is empty, since core assigned to the Executive is considered to be compact in this case.

Damper Usage: General

The static dampers are concerned with the gross allocation problem, i.e., the sets of demands and availables and their relationship. In this context the dampers are used to flag when allocation should be initiated and to gate the execution of selected allocation processes when an external event creates an allocation condition that must be logically preserved until the allocator's response is activated.

The main-level Core Allocator is invoked when the dampers are off. Recall that an allocation pass through the Core-Queue would be futile when the dampers are on since core is compact up to both free and 'swapable' holes. Specific damper usage is described in the allocator phase overviews that follow.

Notice that for any specific allocation case (when a demand or resource has been selected), the dampers are meaningful in deciding whether to attempt the allocation, but they are meaningless within the allocation process itself because they have no bearing on demand/ resource selection.

The last damper comment is a word of caution. Since the dampers directly effect allocator initiation, the damper settings must be carefully maintained. If they are inaccurate the core Allocator will suffer from an excessive number of futile allocation passes or a crippling lack of allocation passes.

Main-Level Allocation

Main-level core allocation includes both demand and swap resource allocation phases. Demand selection in both phases is made from the whole Core-Queue up to the highest priority no-pass.

If both allocation methods are to be initiated, demand allocation always precedes the resource allocation. This means that free core will be filled before any swap-outs are initiated. Main-level allocation phases are so ordered because swapping is considered to be a drastic allocation measure.

Demand Allocation Phase

The demand allocation phase is enabled by the Dispatcher every time it completes its queue service. The demand allocator first queries the Static Core-Map Damper to see if free core is compact. If the damper is on, the demand allocator enables the swap resource phase by passing the processor to it. If the damper is off, free core is <u>not</u> compact; therefore, demand allocation is in order.

Prior to starting demand selection, a Core-Queue selection depth tally is initialized to the Core-Queue depth of the highest priority no-pass if one is present or zero if there is none. This tally is used in demand selection to indicate the logical end of the Core-Queue.

The Core-Queue/Core-Map Fence is then used as a demand selection filter such that a demand is selectable if it crosses (is $\leq$) the fence. Notice that the fence is being used as an upper bound on the largest free core hole. If a demand is selected, the

Core-Map is searched for the best-fit free core-hole. This search is the resource selection.

If no fit is found, the fence is adjusted so that the selected demand does not cross it. An interrupt breath is then taken to allow courtesy-call interruptions and demand selection is restored with the first demand following the previously selected demand.

If a best-fit core-hole is found, the hole is assigned to the demand and the designated Keyword Processor's core load is initiated. During the load the size of the demand is saved in .TPAD lower if its associated TASK since the Core-Map entry built by demand/ resource assignment occupies the same TASK cells that the Core-Queue entry describing the demand used. The demand size is used to build the BAR after the load successfull completes.

Bypass counts of all demands skipped over or preceding the selected demand are then updated. If the update forces a no-pass, the demand allocation phase is terminated and swap resource allocation is enabled; otherwise, an interrupt breath is taken to allow courtesy-call interruptions, followed by the resumption of demand selection.

Demand allocation terminates when either the Core-Queue selection depth tally runs-out or the end of the Core-Queue is detected. Prior to enabling the swap resource phase, the Static Core-Map Damper is set <u>on</u>. This must be done since there are <u>no</u> eligible demands that will fit in any of the remaining free core-holes, i.e., free core is compact. Notice that the Core-Queue/Core-Map fence has been adjusted as a resource upper bound during this allocation phase.

<u>Secondary Controls Associated with Demand Allocation</u>

During initialization the Core Allocator's main-level execution phase cell (.EAIXP) is set to one. This cell is a courtesy-call/main-level interlace control. The upper half of the cell is used by the courtesy-call allocation routines to maintain a negative count of no-pass's either linked to the Core-Queue or forced by bypass count updates during the interruption of the main-level demand allocator. This count is effectively zeroed at the completion of the main-level demand phase. The lower half of this cell contains the main-level execution phase setting. This is zero when the main-level allocator is <u>not</u> enabled, one where main-level demand allocation is enabled, and two when main-level 'swap' resource allocation is enabled.

Whenever a demand is selected, the allocator's Core-Queue position and Core-Queue depth tally are saved in cell .EACQS for resumption of the demand selection. The Core-Queue position is a pointer to the selected demand, in particular the Core-Queue entry itself. The Core-Queue selection depth tally has two possible interpretations. If the tally is positive, it represents the remaining number of demands eligible for selection. This count includes the highest priority no-pass. If the tally is negative, it represents the negative Core-Queue depth of the selected demand.

One other cell set as a result of a successful selection is the main-level demand allocator's current Core-Queue depth (.ENUQD). This is the depth of the selected demand. The depth is found by subtracting the tally in .EACQS from the Core-Queue depth of the highest priority no-pass which is kept in .ENPQD lower. The latter depth is zero if there are no no-pass's. The allocator's Core-Queue depth is used for courtesy-call main-level interlace.

<u>Swap Resource Allocation Phase</u>

The swap resource allocation phase is enabled by the main-level demand allocation phase. The swap allocator first checks the Static Swap Damper to see if core is compact up to swapable holes. If the damper is on, core is compact, so control is returned to the Dispatcher. Otherwise the damper is set <u>on</u> and the swap allocator checks .ESWPE for a nonzero value to determine if there are any swapable core-holes. If there are none, control is returned to the Dispatcher, otherwise, the swap allocator initializes itself.

Prior to searching for a swapable hole, an internal resource selection filter is set to the Core-Queue/Core-Map Fence size. This filter is maintained internally by the swap allocator as a lower bound on the smallest demand so that the fence will remain the best upper bound on the available core resources.

The Dispatcher's-Queue is then searched for the lowest priority Keyword Processor that is eligible for swap. This search is the resource selection since the TASK associated with the swap-eligible Keyword Processor contains the Core-Map entry that describes the swap hole.

When a swap hole has been found, the swap hole size is combined with the size of any adjacent free core-holes. The total size is then compared against the internal resource selection filter to see if it is potentially larger than the smallest demand. If the size is not greater than the filter, resource selection is resumed. Otherwise demand selection commences with the total swap hole size used as a selection filter. The Core-Queue is scanned from high to low priority. All entries up to, and including, the highest priority no-pass are considered in the scan. The demand selection criteria is 'first-fit'.

If no fit is found, the internal resource selection filter is set to the total size of the previously selected swap hole, an interrupt breath is taken to allow courtesy-call interruptions, and resource selection is restarted.

If a fit is found, an attempt to get a free Swap-Map entry and sufficient swap-file space is made. An unsuccessful attempt results in an interrupt breath and the resumption of resource selection. However, if a map entry and space are available, initial resource assignment is made to the selected demand by altering the Core-Map entry describing the swap hole to include the adjacent free core-holes. The true LAL of the swap hole (that is the LAL of the Keyword Processor occupying the swap hole) is saved in .TPAD lower of the TASK describing the swap resource because of the map modification. In addition, a pointer to the replacement TASK is saved in .TPAD upper of the TASK to be swapped-out.

Swap-out of the Keyword Processor occupying the swap hole is then initiated and the bypass counts of those demands which were skipped over are updated. Swap allocation terminates if this bypass count update forces a no-pass. Otherwise an interrupt breath is taken and resource selection is restarted from the previously selected resource in the Dispatcher's-Queue.

Swap resource allocation terminates when either a no-pass is forced or the resource selection search detects the end of the Dispatcher's-Queue. In either case, control passes to the Dispatcher.

Secondary Controls Associated with Swap Resource Allocation

There are two secondary controls affected by main-level swap allocation. The first is the Core Allocator's execution phase cell (.EAIXP). This cell is set to two at the onset of swap resource allocation and zero when this allocation phase is completed.

The other cell affected is the swap resource selection's dispatcher's-Queue position (.ESDQP). Whenever a resource is selected, a pointer to the next higher priority entry in the Dispatcher's-Queue is saved in the upper half of this cell. This pointer is used for resumption of the resource selection's scan of the Dispatcher's-Queue and for courtesy-call/main-level interlace.

Each time a swap resource is selected, demand selection is started at the beginning of the Core-Queue. Interrupt breaths are positioned after demand selection (successful or not), but before resumption of resource selection. Consequently no

Core-Queue position controls are required for courtesy-call/main-level interlace as is the case for demand allocation.

Courtesy-Call Allocation

There are two courtesy-calls in which core allocation can take place. These are the courtesy-calls attached to the Transaction Scheduler's Intercom read to TPE and the Keyword Processor swap-out I/O caused by swap resource allocation.

New Demand Allocation

The first courtesy-call allocation is called 'new demand' allocation. This is concerned with a new transaction's core demand. When the new demand is priority linked into the Core-Queue, its allocation eligibility relative to the Core-Queue is determined. A demand-not-eligible return from the link routine results in the suspension of this allocation attempt. In this case the demand will have to be serviced by the main-level allocator. If the demand is eligible for further allocation consideration it is called 'Q-eligible'. In this case an attempt to allocate core to the new demand must be made because the link routine adjusts the interlace controls under the assumption that such an attempt will be made.

Demand allocation will be attempted if the main-level allocator was interrupted in its demand phase or the Static Core-Map Damper is on and if the new demand crosses the Core-Queue/Core-Map Fence. Allocation is permitted and necessary during interruption of main-level demand allocation because the 'Q-eligibility' of the demand implies that the main-level allocator's queue position is past the new demand. If nothing is done an indefinite delay in allocation to this demand could result. Since the Static Core-Map Damper will be set on at the end of main-level demand allocation phase.

Allocation is permitted when the damper is on because free core is compact and 'Q-eligibility' in this case means the new demand's priority is greater than that of any no-pass present in the Core-Queue. When the damper is off no allocation attempt is made since this damper state implies that a 'favorable' Core-Map change has occurred. This change might be sufficient to allow main-level allocation to a demand whose priority is greater than that of the new demand. Even though bypass logic would allow allocation to the latter demand, it is a sense of 'fair play' that dictates the wait in this case.

Resource selection criteria is best fit. If no available resource fits, the fence is adjusted so that the new demand no longer crosses it. Otherwise having successfully selected a resource, core assignment is made, the transaction designated Keyword Processor's core-load is initiated and the bypass counts of higher-priority demands in the Core-Queue are updated. If a no-pass is forced by the update, the upper half of the main-level allocator's execution phase cell (.EAIXP) is decremented by one. Control is then returned to the Transaction Scheduler.

If demand allocation was not allowed or not successful, swap allocation is considered. The main-level allocator's execution phase cell (.EAIXP) is queried to check if a no-pass was forced or linked within a courtesy-call. If a negative no-pass count exists, the main-level allocator's phase checks are bypassed. Otherwise the main-level's phase is examined as follows.

If the main-level was interrupted in its demand phase, the Static Swap Damper is turned off to force a main-level swap allocation attempt. This is done because swap allocation within the courtesy-call could freeze-up free core-holes, making them inaccessible to the main-level.

If the main-level was interrupted in the swap phase, the position of its 'swap' resource selection within the Dispatcher's-Queue (.ESDQP) is reset to the start of the queue. This is done so all swap holes will be re-examined in light of the new demand.

In either case of main-level allocator interruption, control is returned to the Transaction Scheduler.

At this time the Static Swap Damper is checked to determine if swapable core is compact. If it the damper is off, indicating that it is not compact, allocation is suspended and control returned to the scheduler. Otherwise swap resource selection is initiated for the new demand. Resource selection criteria is first-fit. If a resource is found, swap-out of the Keyword Processor occupying the swap-hole is initiated and bypass counts are updated. If a no-pass is forced, the courtesy-call no-pass count in .EAIXP upper is decremented by one.

Whether or not the swap allocation is successful, control is returned to the Transaction Scheduler.

Swap-Out Allocation

Swap-out courtesy-call allocation is concerned with the free core hole created by a Keyword Processor swap-out as a result of swap resource allocation. This core-hole requires special handling because it can be the sum of non-contiguous free core-holes plus part of the intervening swap-hole.

The major complication occurs when the Keyword Processor's swap-out courtesy-call interrupts the main-level in its demand phase. If the free hole created by the swap-out is larger than any existing hole, it is conceivable that a demand already rejected by main-level demand selection could fit within this hole. If nothing is done, the passed over demand could lose-out to a lower priority demand yet to be examined by main-level demand selection. Even though bypass logic would permit this to happen, it definitely is not the purpose or intent of such logic, particularly since the higher priority demand might fit in the new hole.

This problem can be rectified in two ways. The main-level demand allocator's Core-Queue position could be repositioned via the interlace control. .EACQS to the start of the queue. This would result in all demands being reconsidered in light of the new core-hole. The selected method is to try to allocate the new hole to those demands already rejected or passed over by the main-level. This method was chosen for timing reasons only.

During the swap-out courtesy-call, the LAL of the swap partition is adjusted to the upper address limit (UAL) of the previous (lower address) core partition. This is equivalent to combining the swap partition with the free hole below it. True core assignment to the demand to be swapped-in can then be made. At this time the free hole resulting from the swap-out is known and described by the Core-Map entry built for the demand being loaded. The size of the hole is compared to the Core-Queue/Core-Map Fence to see if it is the largest free core-hole. If the hole size does not cross the fence (>), this indicates that it is not the largest and that core allocation need not be attempted. Otherwise, the Core-Queue is checked to see if any demands are present. If it is empty the fence is set to the size of the free hole and of course no allocation is attempted.

If there are demands, resource allocation will be attempted. In this case, the main-level allocator's execution phase must be queried. If the main-level was interrupted in its demand allocation phase, a demand selection tally is set to the Core-Queue depth of the main-level allocator's demand selection. This tally is used so only those demands already rejected by the main-level will be considered. If the main-level was interrupted in its swap resource phase or not interrupted at all, the demand selection tally is set to the Core-Queue depth of the highest priority no-pass or zero if none. In this case all eligible demands will be considered.

Recalling that the objective is to allocate the swap created hole, demand selection commences. Selection criteria is first-fit. If a fit is found, resource assignment is made, core-load of the demand associated Keyword Processor is initiated, and bypass counts are updated. If a no-pass is forced, the count of courtesy-call no-pass's in .EAIXP is negatively incremented. Notice that the Core-Map entry built by demand resource assignment for the selected demand describes any residue of the free hole that was in excess of this demand.

Resource allocation continues until the demand selection tally runs out or the free core-hole residue no longer crosses the Core-Queue/Core-Map Fence. In the former case, the hole residue is compared to the fence since it could still cross it. If the core residue does cross the fence, then the fence is set to the size of the residue hole.

When resource allocation has terminated or if no allocation was attempted, a check is made to see if the remaining free core-hole lies below (lower address) a swap hole. If it does the total size of the swap hole is computed and compared to the fence. If the total hole size crosses the fence and the main-level allocator was <u>not interrupted or interrupted in the demand phase, the Static</u> Swap Damper is turned off. This will trigger the main-level swap resource phase, which will try to allocate the swap-hole. If the main-level's phase is neither of the above, it must be in its swap resource allocation phase. In this case, the main-level's resource selection position within the Dispatcher's-Queue (.ESDQP) is reset to the beginning of the queue so that all swap holes will be re-selected and examined. A straightforward check of whether the main-level is past the swap hole in question is not possible. This is because the Dispatcher's-Queue is not linked serially and no queue depths are maintained for its entries. Consequently any repositioning must be to the start of the queue.

At this time the swap created hole has been incorporated into the gross allocation state. All that remains is to process the swapped-out Keyword Processor's demand. The associated TASK is linked to the Dispatcher's-Queue at this time. If the TASK is marked for abort or if it is GEWAKE'd, it is left linked to the queue and the courtesy-call is terminated. Otherwise it is unlinked and subsequently linked to the Core-Queue.

If the link routine specifies that the demand is 'Q-eligible', an allocation attempt must be made. This is accomplished by calling the 'new demand' allocation much like the Transaction Scheduler does. Upon return the courtesy-call is ended.

The swapped-out demand is purposely linked to the Core-Queue _after_ the swap created hole has been allocated. This order is followed because a Keyword Processor is generally marked eligible for swap as a result of excessive resource usage.

Core Allocator Entry Points

Symbol	Title
CAL010	Link New TASK to Core-Queue
CAL011	Link Old TASK to Core-Queue
CAL060	Unlink TASK from Core-Queue
CAL090	Main-Level Core Allocator
CAL120	New TASK Allocator
CAL140	Allocate Core Start Load
CAL150	Allocate Core Start Swap
CAL160	Update Bypass Counts
CAL164	Get No-pass Q-depth
CAL170	Initialize Swap-Eligible TASK Search
CAL171	Resume Swap-Eligible TASK Search
CAL180	Load/Swap-in Courtesy-Call
CAL190	Swap-out Courtesy-Call
CAL220	Release Core & Adjust Allocator's Controls
CAL230	Calculate & Check Size of 'Swap' Hole
CAL240	Mark TASK Eligible for Swap
CAL250	Check Swap/Load I/O Status
AI0100	Setup & Do Swap-in I/O
AI0110	Setup & Do Swap-out/Swap-in I/O
AI0120	Setup & Do Startup Rollout

Link to Core-Queue

FUNCTION:

This routine links a TASK via cells .TMEM and .TPRIO to the Core-Queue according to the priority assigned to the TASK in .TPRIO. Affected Core Allocator controls are updated. The routine also determines the allocation elibigility of the newly linked TASK relative to any no-pass present in the Core-Queue and to the main-level Core Allocator, if it was interrupted within its Demand Phase.

ENTRIES

Common calling parameter is:

CP1 - L(TASK) to be linked to the Core-Queue.

EP1 - Link New TASK to Core-Queue (CAL010).

This entry calls an optional routine to assign an initial priority and bypass count in .TPRIO and .TLAL respectively.

EP2 - Link Old TASK to Core-Queue (CAL011).

This entry uses existing values in .TPRIO and .TLAL.

RETURNS:

RR1 - Call+1, Not eligible for allocation.

RR2 - Call+2 Eligible for allocation.

Unlink Core-Q Entry

FUNCTION:

This routine unlinks cells .TMEM and .TPRIO of the designated TASK from the Core-Queue. Core Allocator controls are updated.

ENTRIES:

EP 1 - Unlink Core-Queue Entry (CAL060).

CP1 - L(TASK) to be unlinked from Core-Queue.

RETURNS:

Return is made to the calling routine at Call+1.

Main-Level Core Allocation

FUNCTION:

This routine attempts to allocate core from existing free core-holes (demand allocation) if the Static Core-Map Damper (.EDAMP) is off. The routine also attempts to allocate core from holes that would be created by swapping-out swap-eligible Keyword Processors (swap resource allocation), if the Static Swap Damper (.EDAMP+1) is off. In both cases the Core-Queue is serviced up through the highest priority no-pass, if any, or the end of the queue.

ENTRIES:

EP1 - Main-Level Core Allocator (CAL090).

This entry is called exclusively by the Dispatcher after its queue service.

RETURNS:

Return is always to the Dispatcher.

New TASK Allocation

FUNCTION:

This routine attempts to allocate core to a new TASK received by the Transaction Scheduler or spawned via a Keyword Processor-to-Keyword Processor communication request. This routine assumes that all Core-Queue eligibility criteria have been met by the designated TASK. Both demand and swap are conditionally attempted. This routine generally executes at the courtesy-call level.

ENTRIES:

EP1 - New TASK Allocation (CAL120).

CP1 - L(TASK) to allocate core to.

RETURNS:

Return is made to the Call+1.

CORE ALLOCATOR

Allocate Core Start Load

FUNCTION:

This routine unlinks the designated TASK from the Core-Queue and allocates core to the TASK by building a Core-Map entry in .TMEM and .TLAL. Core is normally allocated at the top of the specified hole except when the hole is above (higher core address) a swap-eligible TASK. In the latter case, core is allocated at the bottom of the hole. Lastly, the routine starts the necessary load/swap-in I/O.

ENTRIES:

EP1 - Allocate Core Start Load (CAL140).

CP1 - L(TASK) to which core is to be allocated to.

CP2 - L(Core-Map entry) which holds the core-hole to be allocated.

RETURNS:

Return is made to the Call+1.

Allocate Core Start Swap

FUNCTION:

This routine attempts to get Swap-File space and a free Swap-Map entry in which to reserve the space for the designated swap-eligible TASK. If successful, the free core-hole lying above the swap-eligible Keyword Processor is combined with the core partition described by the swap-eligible TASK's Core-Map entry and the swap-out of the swap-eligible Keyword Processor is initiated.

ENTRIES:

EP1 - Allocate Core Start Swap (CAL150).

CP1 - L(TASK) to be swapped-out.

CP2 - L(TASK) to be swapped-in.

RETURNS:

RR1 - Call+1, Insufficient swap-file space to start swap.

RR2 - Call+2, Swap-out started.

Update Bypass Counts

FUNCTION:

This routine decrements the bypass count of each TASK in the Core-Queue starting with the TASK pointed to by .TPAD upper of the designated TASK and continuing to the start of the Core-Queue. No-pass TASKS (TASKS whose bypass counts have runout) that are forced by the update are linked via the upper half of their respective .TPAD cells in high to low priority sequence. The base of the no-pass chain is recorded in .ENPQD. Lastly, the Core-Queue depth of the highest priority no-pass is calculated and saved in .ENPQD lower.

ENTRIES:

EP1 - Update Bypass Counts (CAL160).

CP1 - L(TASK) holding backwards Core-Q pointer for update.

EP2 - Get No-pass Q-Depth (CAL164).

No calling parameters.

RETURNS:

RR1 - Call+1, No-pass present in .ENPQD.

RR2 - Call+2, Bypass.

For EP1, a new higher priority no-pass was not forced by this bypass count update, but an old no-pass might be in effect.

For EP2, no no-pass is in effect.

Swap-Eligible TASK Search

FUNCTION:

This routine searches the Dispatcher's-Queue for a TASK that is flagged as eligible to be swapped in its .TFLAG cell. This condition is indicatd if the B.TESW bit flag is on. If a TASK is found, the number of free core blocks (1024 words), which would be freed up if the TASK were swapped-out, is calculated.*

ENTRIES:

EP1 - Initialize Swap-Eligible Search (CAL170).

This entry initializes the search relative to the current disposition of the Dispatcher's-Queue.

No calling parameters.

EP2 - Continue Swap-Eligible Search (CAL171).

This entry resumes the search using the last position saved in Communication Region cell .ESDQP.

No calling parameters.

* The number of freed-up core blocks is the sum of the current core partition size and the size of the previous available core-hole.

RETURNS:

RR1 - Call+1, No swap-eligible TASKs.

<u>Swap-Eligible TASK Search</u>

RETURNS:

RR2 - Call+2, Swap-eligible TASK found.

RP1 - L(Swap-eligible TASK).

RP2 - Number of core blocks that would be freed-up by a swap.

Load/Swap-in Courtesy-Call

FUNCTION:

This routine initializes a new or previously swapped Keyword Processor after it has been loaded into core. If necessary, swap-file space is released. The BAR and other controls are constructed if the load was successful and the TASK is linked to the Dispatcher's-Queue.

ENTRIES:

This routine is Courtesy-Call driven.

RETURNS:

Return is made to GCOS via a MME GEENDC.

Swap-out Courtesy-Call

FUNCTION:

This routine deallocates core from the swapped-out TASK, combines the freed-up core with any adjacent available core and allocates the total to the TASK to be swapped-in. Swap-in is then initiated and the swapped-out TASK is unlinked from the Dispatcher's-Queue and linked to the Core-Queue, if it is not marked for abort. Lastly, an attempt is made to allocate the core-hole resulting from the swap. If the main-level Core Allocator was interrupted during its demand (load) allocation phase, only those TASKs already serviced by the main-level allocator are considered.

ENTRIES:

This routine is Courtesy-Call driven.

RETURNS:

Return is made to GCOS via a MME GEENDC.

Release Core & Adjust Allocator's Controls

FUNCTION:

This routine releases all core assigned to the designated TASK. It is subsequently adjusts the Core Allocator's Core-Queue/Core-Map fence (.EQMIN) and Static dampers (.EDAMP) as required.

ENTRIES:

EP1 - Release Core & Adjust Allocator's Controls (CAL220).

CP1 - L(TASK) from which core is to be released.

RETURNS:

Return is always passed to the Call+1.

Calculate & Check Size of 'Swap' Hole

FUNCTION:

This routine calculates the number of contiguous core blocks that would be freed-up if the designated TASK were swapped-out. (Core-holes above and below the designated TASK are combined with the core space assigned to the TASK to get the resulting hole size). The size of the 'Swap' hole is then compared to .EQMIN, the Core-Queue/Core-Map Fence.

ENTRIES:

EP1 - Calculate & Check Size of 'Swap' Hole (CAL230).

CP1 - L(Swap-eligible TASK).

RETURNS:

RR1 - Cal+1, 'Swap' hole size <=.EQMIN.

RR2 - Call+2, 'Swap' hole size > .EQMIN.

Mark TASK Eligible for Swap

FUNCTION:

This routine is intended as a one-stop call to flag a TASK as eligible for swap and to adjust any affected Core Allocator controls. Convenience aside, this routine exists to minimize code that adjusts the Core Allocator's Dampers.

Caution: Once a TASK is marked eligible for swap, inhibited code must be used to prevent the swap from possibly happening at the courtesy-call level. Therefore, either (1) all processing that requires the TASK to be in-core should be done before calling this routine or (2) inhibited code should be employed during such processing after return from this routine.

ENTRIES:

EP1 - Mark TASK Eligible for Swap (CAL240).

CP1 - Pointer to TASK eligible for swap.

RETURNS:

Return is always made to call+1.

RP1 - Size of the 'swap' hole that would be created by swapping-out the designated TASK.

Check Swap/Load I/O Status

FUNCTION:

This routine checks the major status code for all Keyword Processor swap/load I/O. If the status is bad, the affected TASK is flagged for abort and the appropriate error code is inserted into the TASK.

ENTRIES:

EP1 - Check Swap/Load I/O Status (CAL250).

CP1 - Code to identify the calling routine (currently either the Load/Swap-in Courtesy-Call or the Swap-out Courtesy-Call).

CP2 - L(TASK).

CP3 - Core-LAL of the associated Keyword Processor.

RETURNS:

Return is always to the Call+1.

Check Swap/Load I/O Status

FUNCTION:

This routine checks the major status code for all Keyword Processor swap/load I/O. If the status is bad, the affected TASK is flagged for abort and the appropriate error code is inserted into the TASK.

ENTRIES:

EP1 - Check Swap/Load I/O Status (CAL250).

CP1 - Code to identify the calling routine (currently either the Load/Swap-in Courtesy-Call or the Swap-out Courtesy-Call).

CP2 - L(TASK).

CP3 - Core LAL of the associated Keyword Processor.

RETURNS:

Return is always to the Call+1.

Do Core Allocator I/O

FUNCTION

This routine performs disc I/O for the Core Allocator by building the I/O Select- Sequence and necessary DCWs, depending on the entry point. A pointer to the appropriate courtesy-call routine is inserted into the Courtesy-Call Vector (.TCCV) in the TASK.

ENTRIES:

EP1 - Swap-In (AIO100).

CP1 - L(TASK) to be swapped-in.

EP2 - Swap-Out/Swap-in (AIO110).

CP1 - L(TASK) to be swapped-out.

CP2 - L(TASK) to be swapped-in.

EP3 - Startup Roll-Out (AIO120).

CP1 - L(Pseudo TASK) used by Initialization Routine.

RETURNS:

Return is always to the Call+1.

DISPATCHER

Function

The Dispatcher functions primarily to select eligible TASKs for dispatch on a priority basis; to monitor the status of system functions, gross Executive resource utilization and TASK system requests; and to enable system functions and services.

Introduction

The Dispatcher is the control focal point of the Executive because control is always passed or returned to it. As a result it is responsible for monitoring all other internal Executive functions and for periodically or reflexly enabling them. Through its queue all TASKs eligible for dispatch or requiring a system service are known.

The normal sequence of events within the Dispatcher is to service its queue; conditionally enable system functions; and select, then dispatch to an eligible Keyword Processor.

Dispatcher's Queue

All TASKs eligible for dispatch or requiring an Executive service are linked together to form the Dispatcher's-Queue. A queue-entry proper consists of TASK cells .TPRIO, .TFLAG and .TFLAG+1. The queue is forward and backward linked with pointers to the head and tail of the queue along with the number of TASKs linked to the queue maintained in Communication Cell .EDSPQ.

The TASKs are chained together according to the priorities assigned in the lower half of their .TPRIO cells. When following the forward queue pointers in the upper half of the queue-entry (.TPRIO cells), the priority sequence is ascending, and when following the backward queue pointers in the upper half of the queue-entry (.TFLAG cells) the priority sequence is descending.

Queue entry cells .TFLAG lower and .TFLAG+1 lower contain attribute and status bit flags. Attribute flags define the characteristics of the Keyword Processor designated by TASK cell .TSID. These flags are copied from the Keyword Processor Profile when the transaction is first received by the Executive. Status flags define the current processing state of the Keyword Processor and which queues the TASK is linked to.

The bit flags in .TFLAG lower are further labeled 'Q-scanned' flags. This label serves to differentiate between those flags used by the Dispatcher's-Queue Service and Dispatch Select, and those that aren't. This grouping of the bit flags is a consequence of the method used to scan the queue.

Specifically, the .TPRIO cell of all TASKs is assembled with an even offset from the origin of the TASK. Furthermore the space assigned for TASK construction in the Executive has zero origin mod 2. As a result, TASK cells .TPRIO and .TFLAG form an even word pair since .TFLAG always follows .TPRIO. This is done to allow logical operations on the bit flags in .TFLAG lower to be executed with a minimum of instructions when following either the forward or backward Dispatcher's-Queue pointers.

Bit flags that are to be used to 'single-out' a TASK when examining the queue as a whole are placed in .TFLAG lower. Since these flags are used when the queue is processed or searched, they are called 'Q-scanned'. Bit flags that are queried relative to a known TASK or queue-entry are placed in .TFLAG+1.

Notice that TASK cell .TPRIO is ued both in a Dispatcher's-Queue entry and a Core-Queue entry. Consequently, a TASK cannot be linked to both queues simultaneously.

Dispatcher's-Queue Service: Introduction

Some Executive services and functions, which are directly identifiable with a specific TASK, cannot be initiated when their need is indicated or cannot complete their function when given control. For example, TASK termination cannot be initiated within courtesy-call and requests for buffer space might be unsuccessful. When these service conditions occur, it is necessary to queue the request in order that the function can be either initiated or restarted, whichever the case.

A queue is not defined for each such service since the necessary queue-entry information would be essentially the same; thus independent of the particular service. A generic queue-entry would contain a pointer to the TASK that was being processed and the IC&I where the service is to be given control.

Only one queue is required and this queue is embedded as an integral part of the Dispatcher's-Queue. Relative to this queue, the only queue-entry parameter is the service IC&I in which the queue-entry lies and thereby identifies the TASK being processed by the service.

The Dispatcher's-Queue Service detects and enables the TASK associated service requests as dictated by its queue.

Dispatcher's Dynamic Queue Service Mask

The service mask is used to select those queue entries within the Dispatcher's-Queue that are waiting to be serviced. The mask is positioned at symbolic location .EDQSM within the Communication Region.

The service mask is the logical sum (OR) of those 'Q-scanned' bits which flag a service request. Notice that this can be formed as a binary sum since the applicable bit flag definitions (B.XXXX) do not overlap.

During the Dispatcher's-Queue Service, the mask is serially 'ANDed" to the "Q-scanned' bits in .TFLAG lower of each queue-entry. If the product is non zero, one or more of the 'Q-scanned' bits being queried by the mask has been set on (is one). This means the queue-entry under examination is waiting for a service.

The service bit mask can be dynamically altered to allow the conditional selection of some services. The idea here is to let the mask reflect an internal 'go/nogo' service status so that when a service <u>cannot</u> be provided, the waiting queue entries are not detected (ignored) in the queue scan.

Not all services can be summarized so succiently. However, it is the nature of some services that allows the determination of whether future requests can be satisfied. These services effectively disable themselves when they can no longer accomplish their function. In these cases, there is a complementary system function for each service that determines when the service can be enabled.

An example of such a service is the processing of Intercom-Queue entries. When an available entry is requested and there is only one remaining, it is clear that once this entry is assigned, future queue requests would be fruitless. Whenever this occurs, the service would suspend itself. When a queue-entry is released by the complementary service and it is the only queue-entry available, queue requests would again be enabled.

All services that can be dynamically enabled/disabled via the service mask must be assigned dedicated 'Q-scanned' bit flags. The remaining services that are not so conditioned could be lumped together into one bit flag. This is seldom done, however, since the 'Q-scanned' bits also serve to describe the state of the Keyword Processor represented by the queue-entry.

Recognition of a particular service's request is disabled by 'ANDing' the Dispatcher's-Queue Service Mask with the 1's-complement of the services's related 'Q-scanned' bit flag. That is setting the appropriate bit off in the mask. Recognition is enabled by simply 'ORing' the 'Q-scanned' bit flag back on in the service mask.

Queue Service: When and How

The Dispatcher's-Queue Service is enabled whenever a Keyword Processor timer runout occurs or an initial system service or function request cannot complete or does not return to the Keyword Processor. When initiated, the lower half of the Dispatcher's-Queue Service Position cell (.EDQSP) in the Communication Region is set to a non-zero value. This halfword is used to indicate that the queue service has been activated and is in progress.

At this time the TASK Alarm Clock (.ETACK) is checked to set if it has been set, that is not -1. If so the time of day is obtained and saved in Communication Region cell .ETIME. If the TASK Alarm Clock is ringing (.ETACK $\leq$ .ETIME), the Dispatcher's-Queue Service Mask is adjusted so that GWAKE'd TASKs linked to the Dispatcher's-Queue will be detected. Actual scan of the queue is then initiated, starting with the highest priority entry or TASK linked to it. The scan is continued via the backward queue pointers in the upper half of the queue-entry .TFLAG cells. The queue service mask is used, as explained earlier, to recognize those entries awaiting service.

When an entry is found, a pointer to the next queue-entry is saved in cell .EDQSP upper for later resumption of the scan. This cell is also necessary to protect the queue service from getting 'lost', since the Dispatcher or the service routine it enables might be interrupted by a courtesy-call that links or unlinks an entry to the Dispatcher's-Queue. In this context .EDQSP functions as a main-level/courtesy-call interlace control.

The queue service is terminated when the scan detects a zero backward Dispatcher's-Queue pointer, or resumption of the queue service finds .EDQSP upper is zero. The latter case occurs if the lowest priority queue-entry was last recognized for service so that the next backward queue-entry pointer, saved in .EDQSP, would be zero.

<u>Service Initiation/Restart</u>

The Dispatcher's-Queue does <u>not</u> attempt to determine what service is needed by the selected queue entry from the "Q-scanned' bits set on in the entry. The basic reason for this design approach is twofold: (1) the system service or function is the logical place wherein internal and service related TASK status should be examined and the necessary processing course plotted when the functions cannot be initiated or completed, since all necessary information is known at that point; and (2) the Dispatcher's design remains open-ended with this modular service approach. As a result the Dispatcher is not burdened with having to arrange and classify the different bit flag combinations that may be present in the queue-entry.

A service vector in .TFLAG+1 upper of the entry is used to enable the processing for a selected entry. The vector is the IC of where the service is to be given control. When the Dispatcher has detected a queue-entry waiting for service, it saves the pointer to the associated TASK and checks the service vector to make sure it is not zero. If zero, the TASK is aborted; otherwise the IC is saved and the service vector is zeroed. At this time the service is enabled.

Only the TASK pointer is currently supplied for the service, no other cells set by a dispatch are supported.

Service Considerations and Restrictions

When a service is unable to complete its function, it is the service's responsibility to:

(1) Ensure that the associated TASK is linked to the Dispatcher's-Queue,

(2) Set the appropriate 'Q-scanned' bit flag on in the TASK so the service can be restarted, and

(3) Set the service vector in .TFLAG+1 to point to the location where the service wishes control to be returned when it is later enabled to attempt to complete its function.

There are two restrictions that must be observed by the service functions when it is necessary to use the Dispatcher's-Queue Service. Only one level of IC depth is allowed the service when it disables itself. This means that if control is one or more levels removed from the service routine proper when the determination of service dysfunction is made, control must be returned up to a point where one IC defines the service's position, i.e., to where the service becomes reusable. This problem is currently unavoidable with in the Executive's modular structure since there is no service IC stack within the TASK.

The other restriction applies to data storage for the service routine when it cannot complete. The only storage area allowed the service is the TASK space from .TPAD+1 through the end of the TASK. The size of this area is variable depending on the values supplied the .TASK macro during the Executive's assembly. All TASK or Keyword Processor related data must be saved in this area, however it is intended that the amount of data be a minimum. If a service is frequently requested and its dysfunction judged to be likely, consideration should be given to defining additional TASK cells to hold the necessary information.

When the service is enabled by the Dispatcher's-Queue Service, it is the service's responsibility to turn off the "Q-scanned' bit which was used to flag a 'need service' state within the designated TASK. Failure to do so could result in a subsequent attempt by the queue service to process the original request. In this case, the service vector would be zero from the original enable so the TASK would be aborted.

Service Returns

Service routines that cannot complete their function or do not return control to a Keyword Processor must return to the Dispatcher at symbolic location DSP102. When this occurs, the Dispatcher's-Queue Service is either initiated or resumed.

Service routines that can be initiated or restarted by the queue service and that successfully complete their function must return to DSP302 if they return control to a Keyword Processor. When such a return occurs, the Dispatcher's-Queue Service Position (.EDQSP) is checked to see if the queue service is enabled. If so, the service is restarted where it left off, otherwise a GELBAR is reissued to the Keyword Processor that was serviced. Thus a GELBAR or dispatch is never issued to a successfully serviced Keyword Processor when the service was enabled by the Dispatcher's-Queue Service.

System Function Enables

When the Dispatcher has completed its queue service, it enable the Transaction Scheduler , Core

Allocator and Remote I/O Supervisor

The Transaction Scheduler is enabled both to issue the first Intercom read to TPE at startup time and to restart the scheduler in the event it has stalled-out. Recall that once the scheduler is enabled, it continues to request transactions from TPE at the courtesy-call level until it cannot obtain a free TASK or sufficient input buffer space, both being required for each read that it issues. When this occurs the Scheduler is said to have stalled-out since it must terminate its courtesy-call without issuing an I/O.

Normally the Transaction Scheduler ignores the enable by simply returning control to the Dispatcher. However, if the scheduler has stalled-out as indicated by communications cell .ESTAL, the enable causes the scheduler to try to restart itself.

The Core Allocator treats the Dispatcher's enable in much the same fashion. That is, the enable may result in an allocation attempt or it may be ignored. Decisions in this case are based on the static dampers at Communication Region cell .EDAMP.

The Remote I/O Supervisor uses its enable to perform line service functions which cannot be event driven or initiated. This includes new connect processing and 'sleep'/time disconnecting processing.

Dispatch-Select

The function of Dispatch-Select is to select that highest priority TASK linked to the Dispatcher's-Queue whose Keyword Processor can make immediate use of the processor. A Keyword Processor can gain control _only by being selected_.

In general, dispatching select logic is a two phase process. The phases are the priority selection of a TASK followed by a determination of processor eligibility for the chosen TASK. The select procedure consists of first selecting a TASK from the Dispatcher's-Queue according to a priority algorithum. Having chosen a priority-eligible TASK, a test is then made to see if the TASK related Keyword Processor can make immediate use of the processor. If so, a dispatch can be paid to the Keyword Processor; otherwise the select procedure must be repeated, this time selecting the next highest priority TASK linked to the queue.

Within the Executive, Dispatch-Select priority logic is priority pre-emptive. This choice results in the priority selection being implicitly accomplished by processing the Dispatcher's-Queue from high to low priority, i.e., by following the backward queue pointers. Thus whenever Dispatch-Select is initiated, priority TASK selection always starts with the highest priority TASK linked to the queue, even if it was the last TASK that was dispatched to.

Once a TASK has been priority selected, its processor eligibility is determined with the Dispatcher's Select-Eligibility Mask. This mask functions identically to the Dispatcher's-Queue Service Mask. The mask is applied to the 'Q-scanned' bits in .TFLAG of the selected queue-entry. This is the second use of the "Q-scanned" bits. In this context the applicable bit flags are used to indicate those Keyword Processor conditions which make it ineligible for processor assignment.

Actual Dispatch-Select is accomplished by scanning the Dispatcher's-Queue from high to low priority using the Dispatcher's Select-Eligibility Mask. Within this scan, a queue-entry is selectable for dispatch if the logical product of the mask and the entry's 'Q-scanned' bits is zero.

Dispatch

A Keyword Processor dispatch can result from a Dispatch Select, an I/O interrupt that breaks a dispatch or the successful completion of an initial fault service request.

DISPATCHER

Dispatch to a Keyword Processor is effected by a MME GELBAR. As such, a dispatch consists of:

(1) Adjusting the BAR to the Keyword Processor's memory bounds,

(2) Loading the timer register with the dispatch time quantum,

(3) Restoring the processor registers and EIS address registers, pointer and length registers as last set by the Keyword Processor, and

(4) Transferring control to the Keyword Processor at its last interrupted location.

All information required by the GELBAR is maintained in TASK cells .TBAR, .TICI and .TTIMQ except for the processor register values which are safe-stored at symbolic location .KREG in the Keyword Processor's Prefix and in TASK areas .TAREG and .TEPL. Notice that a pointer to .KREG is maintained in the lower half of .TBAR.

The GELBAR time quantum is set to a literal constant of 64 ms within the Dispatcher except when cell .TTIMQ of the selected TASK is <u>not</u> zero. In the latter case the time quantum is set to the time remaining from an earlier dispatch as saved in .TTIMQ.

Dispatching with variable time quantums of less than 64ms (for uniprocessor system) is currently not effective and not done within the Executive. The reason is that .MFALT does not differentiate between a GELBAR timer runout (TRO) and a normal TRO. In either case, the Executive will be taken out of execution and the processor reassigned. Consequently, any time remaining from the GCOS dispatch to the Executive is lost.

Executive communication cells .EBAS5, .ELDSP, .ETIMQ and .ETREG are set at each dispatch. These cells identify the Keyword Processor dispatched to and facilitate fault and interrupt handling.

Lockup Logic

The purpose of Lockup Logic is to monitor the Executives gross state, when it is unable to find anything to do even though the Dispatcher's-Queue is not empty, and to prevent it from slipping into a comatose state of successive relinquishes. Lockups are classified as temporary, resource and unknown.

A temporary lockup is one that normal processing will remove, thus it is considered an acceptable Executive state. As an example, such a lockup could be caused by one TASK, linked to the Dispatcher's Queue, that is waiting for the completion of device I/O. This type of lockup always results in the Executive relinquishing control.

A resource lockup represents a deadly embrace among two or more Keyword Processor over some Executive allocated resource. This type of lockup will not be removed by normal processing. Currently resource lockup can occur because (1) available output buffer space is exhausted and output Intercom is not executing to flush the buffer, thus deadlocking those Keyword Processors requesting additional buffer space; or (2) TASK space is exhausted thus deadlocking the Transaction Scheduler and those Keyword Processors requesting spawn TASKs. Resource lockup can possibly be removed by serially aborting those Keyword Processors in the embrace.

Unknown lockup is a catch all. That is, if the Executive is dead and it can not be determined why, the lockup is unknown. In this event, the Executive is ungraciously aborted by the Dispatcher.

Prior to attempting to identify the nature of the lockup, the accumulated status of all TASKs linked to the Dispatcher's-Queue is formed by 'ORing' together their "Q-scanned'; bit flags. The logical sum is then used for lockup threshold tests.

If the lockup is temporary the Executive relinquishes control. Otherwise it is assumed that a lockup threshold exists and the resource lockup flag (.ERLUF) is examined for a nonzero value. The flag is used to indicate, by a nonzero value, that there has been two consecutive unsuccessful Dispatch-Select attempts and consequently, two consecutive enables of lockup logic. If the flag is zero, it is set on and control is passed to the Dispatcher's-Queue Service for a one-pass attempt to unlock the threshold, If it is successful and the subsequent Dispatch Select finds an eligible TASK, the resource lockup flag is turned off.

If the flag is non-zero, an attempt to determine the threshold type is made from the accumulated TASK status. If the threshold is identified, thw lowest priority TASK in the deadly embrace is aborted. When the TASK termination is complete, control is returned to the Dispatcher's-Queue service. This queue service pass will determine if sufficient resoure was freed-up to make a Dispatch Select possible. If not, lockup logic will again be enabled and the lowest priority TASK of those remaining will be aborted. This sequence of events could continue until all TASKs have been aborted in which case the threshold would have been ineffectively removed.

If a known threshold cannot be determined or does not exist, the lockup is unknown and the Executive is aborted by the Dispatcher.

Dispatcher Entry Points

Symbol	Title
DSP101	Dispatch After GELBAR Timer Run-Out
DSP102	Service Dispatcher's-Queue
DSP301	Startup
DSP302	Reissue GELBAR After Service
DSP303	Reissue GELBAR After Fault
DSP304	Reissue GELBAR After Fault (With Time Quantum)
DSP305	Reissue GELBAR After I/O Interrupt
DSP600	Link TASK to Dispatcher's-Queue
DSP800	Unlink TASK From Dispatcher's-Queue

Dispatcher Entry Points

The Dispatcher allows nine entry points, none of which return control to the original routine that transferred control to it.

ENTRIES:

EP1 - Startup (DSP301).

Control is transferred to this entry point at the completion of initialization.

EP2 - Dispatch after GELBAR TRO (DSP101).

Control is transferred to this entry point whenever a timer runout occurs within a GELBAR. TASK and Dispatcher's-Queue accounting are performed via this entry point.

CP1 - PTR to faulting TASK.

EP3 - Service Dispatcher's-Queue (DSP102).

Control is transferred to this entry point by abort/termination routines and by Keyword Processor service routines that cannot complete the requested service. This entry either initiates or resumes the Dispatcher's-Queue Service depending on .EDQSP.

EP4 - Reissue GELBAR after Service (DSP302).

Control is transferred to this entry point by Keyword Processor service routines that complete the requested service. If the service was enabled by the Dispatcher's-Queue Servcice, control is routed to resume the queue service; otherwise, a GELBAR is reissued to the serviced Keyword Processor. This entry should only be used by services that can be initiated or restarted by the Dispatcher's-Queue Service.

CP1 - Pointer to serviced TASK.

EP5 - Reissue GELBAR after Fault (DSP303).

Control is transferred to this entry point by services that cannot be initiated or restarted by the Dispatcher's-Queue Service. This entry reissues the GELBAR after a Keyword Processor fault or service request has been successfully serviced.

CP1 - Pointer to TASK.

EP6 - Reissue GELBAR after Fault (DSP304).

This entry is identical to EP5, except the GELBAR time quantum must be specified for this entry point.

CP1 - Pointer to TASK.

CP2 - GELBAR time quantum.

EP6 - Reissue GELBAR after I/O Interrupt (DSP305).

Control is transferred to this entry point by the Fault Handler whenever a GELBAR is broken by an I/O interrupt. The Dispatcher reissues the GELBAR with the time set to the time remaining from the original dispatch.

CP1 - Pointer to TASK.

CP2 - Time cuantum remaining from last GELBAR.

AD-A044 612 HONEYWELL INFORMATION SYSTEMS INC MCLEAN VA FEDERAL --ETC F/G 9/2
TRANSACTION PROCESSING OPERATING SYSTEM (TPOS).(U)
AUG 77 R EWING, J BIELSKI F30602-76-C-0277
UNCLASSIFIED RADC-TR-77-276 NL

3 OF 3
ADA044612

END
DATE
FILMED
10-77
DDC

Link TASK to Dispatcher's-Queue

FUNCTION:

This routine links a TASK into the Dispatcher's-Queue using the TASK priority present in TASK cell .TPRIO lower.

ENTRIES:

EP1 - Link Task to Dispatcher's-Queue (DSP600).

CP1 - Pointer to TASK to link.

RETURNS:

Return is always to Call+1.

Unlink TASK from Dispatcher's-Queue

FUNCTION:

This routine unlinks a TASK from the Dispatcher's-Queue and updates TASK Status Flags and Dispatcher's-Queue controls.

ENTRIES:

EP1 - Unlink TASK from Dispatcher's-Queue (DSP800).

CP1 - Pointer to TASK to unlink.

RETURNS:

Return is always to Call+1.

Select TASK for Dispatch

FUNCTION:

This routine searches the Dispatcher's-Queue for the highest priority TASK that is eligible to be dispatched to. Eligibility is determined by checking the status flags in TASK cell .TFLAG lower. Currently, a TASK is selectable if the following status flags are off:

- Device I/O
- Swapping-Out
- In abort
- In Term
- Need Abort
- Need Output Buffer Space
- Need Output Intercom-Queue Entry
- Need Spawn TASK
- Need Wake-Up

ENTRIES:

EP1 - Select TASK for Dispatch (DSPSEL).

No Calling parameters.

RETURNS:

RR1 - Call+1, No eligible TASKs.

RR2 - Call+2, Eligible TASK found.

RP1 - Pointer to cell .Tflag of the selected TASK.

TASK Terminator Function

The TASK terminator functions to release allocated resources back to the Executive, reassign a reusable Keyword Processor, queue Executive messages to the user or TPE and to complete wrapup measurements.

The terminator is called for both normal and abnormal TASK terminations and always returns control to the Dispatcher. It is written to execute at the main-level only.

Abnormal Termination

Abnormal termination can be called not only to abort an in-core Keyword Processor, but also to abort a TASK whose Keyword Processor has been swapped-out or possibly never loaded into core.

Assigned buffer resources are first released by removing the input transaction from the Input Buffer, if present or removing all output message segments from the Output Buffer, if any. TASK cell .TMSG2 is checked to see if it has any queue entries linked to the Output Intercom Queue. If so, the queue is searched to find those entries which specify the abort TASK as the originating TASK. Each such queue-entry is unlinked from the queue and the Output Buffer space assigned to the queue-entry-defined message segments is released.

TASK cell .TSPWN is examined to determine if the TASK belongs to a spawn TASK chain. If it does, all other TASKs linked to the chain are flagged for abort and linked to the Dispatcher's-Queue, if necessary. Notice that TASKs linked to a spawn chain cannot be identified by a unique transaction number, i.e., all chained TASKs specify the same transaction number so that this number only identifies the chain as a whole and not the individual TASKs. As a result, spawn chain processing dictates that the spawn chain itself must be aborted if a chain member aborts.

Abnormal termination processing is complete at this point and the remaining termination is accomplished by the normal termination phase. However, all device I/O or Keyword Processor load/swap-out I/O must be allowed to complete before the normal termination can be initiated. If any I/O is outstanding, the TASK is linked to the Dispatcher's-Queue and the TASK Service Vector is set for normal termination so that the processing can be completed later.

Normal Termination

Normal Termination assumes the terminating TASK's Keyword Processor is in-core, unless the TASK was received from abnormal termination. For a 'true' normal termination, a check is first made to see if the Keyword Processor is 'reusable'. This means that the Keyword Processor can be assigned to a new TASK for subsequent processing of the latter's transaction.

If it is, the terminating TASK's Keyword Processor Profile is located and examined to see if there are any new TASKs awaiting core allocation; i.e., linked to the Core- Queue. If there are none, the TASK is treated as if its Keyword Processor was not reusable. Otherwise, the Core-Queue is searched from high to low priority to find those TASKs which identify the same profile as the terminating TASKs. When one is found, its status flags are checked to see if it is a new TASK. If it is an old TASK it is discarded, otherwise its allocation eligibility relative to the highest priority no-pass (see Core Allocator discussion) is determined. A TASK which is not eligible, stops the search and the terminating TASK is again treated as not reusable. However, an eligible TASK results in the assignment of the terminating TASKs Keyword Processor to the new TASK. Affected Core Allocator controls are updated since this assignment represents core allocation to the new TASK.

If the terminating TASK is not reusable, checks are made to see if it is linked to the Core-Queue or its Keyword Processor is in-core or swapped-out. The TASK is conditionally unlinked from the queue or its assigned core/swap-file space is released.

A check is then applied to all terminating TASKs to determine if they belong to a spawn TASK chain. If not, an Executive message is queued for Intercom output to TPE and possible the user. This message can be an error message or just an output message header specifying an End-of-Transaction status for the transaction number assigned to the terminating TASK (see Intercom I/O Handler discussion).

Lastly, the TASK is unlinked from the Dispatcher's-Queue and the TASK space is released.

TASK TERMINATOR

TASK Terminator Entry Points

Symbol	Title
TRM100	Start TASK abort
TRM370	Restart TASk Abort
TRM400	Start TASK Termination
TRM720	Restart TASK Termination (Need Output Buffer Space)
TRM740	Restart TASK Termination (Need Output Intercom-Queue Entry)
EXM100	Executive Message Intercom
ETX100	Convert & Edit Transaction Number

Terminator Entry Points

The Terminator handles both normal and abnormal TASK terminations. The Terminator allows five entry points all of which ultimely transfer control to the Dispatcher's-Queue Service.

ENTRIES:

Common calling parameters for all entry points are:

CP1 - L(TASK) to be aborted or terminated.

EP1 - Start TASK abort (TRM100).

Control is transferred to this entry point when the designated TASK is to be aborted.

Cell .TERRM in the TASK must hold the error code.

EP2 - Restart TASK Abort (TRM370).

Control is transferred to this entry point by the Dispatcher's-Queue Service when a previous attempt to abort a TASK could not be completed.

EP3 - Start TASK Termination (TRM400).

Control is transferred to this entry point when a Keyword Processor has successfully completed its processing of a transaction. Core and swap-file space are released if the Keyword Processor is not reusable or there are no outstanding messages requiring the Keyword Processor. A EOT message is sent to TPE and the TASK is unlinked from the necessary queues with optional accounting.

EP4 - Restart TASK Termination - Need Output Buffer Space (TRM720).

Control is transferred to this entry point by the Dispatcher's-Queue Service when prior attempts to terminate a TASK were roadblocked by insufficient Output Buffer space.

EP5 - Restart TASK Termination - Need Output Intercom-Queue Entry (TRM740).

Control is transferred to this entry point by the Dispatcher's-Queue Service when prior attempts to terminate a TASK were roadblocked by lack of an Output Intercom-Queue Entry.

RETURNS:

Return is always to the Dispatcher's-Queue Service

OUTPUT INTERCOM PROCESSOR

Initiate Output Intercom

The Executive's Output Intercom Processor is responsible for controlling and sending all Intercom output that is to be written to TPE. The output processor is functionally interfaced to the Executive by means of the Output Intercom-Queue. Its component routines issue the 'true' Intercom output requests necessary to transfer the waiting messages to TPE.

Introduction

The Output Intercom Processor executes at the courtesy-call level once it has been enabled. It remains enabled at this level as long as there is an output message request waiting in the Output Intercom-Queue. When the queue has been emptied, the output processor disables itself. This mode of operation results in the asynchronous execution of the Output Intercom Processor and the remainder of the Executive.

Output Message Request

When an Intercom output message is ready to be written, it must be passed to the Output intercom Processor via the Output Intercom-Queue. This is done by first requesting an available queue-entry. Unassigned queue entries are managed by communication cell .ECOMQ.

In the event that there are no free entries, the request must be periodically repeated until satisfied. The Dispatcher's-Queue service is the normal means used to enable or restart a request. This procedure allows any other Executive processing to be acted on in the interim.

Once a free queue-entry can be assigned to the message, the entry is filled-in with message specifics. In particular, the queue-entry holds a pointer to the first segment of the message, the priority assigned to the message and the location of the originating TASK, if one. All control information regarding this message is deleted from the originating TASK after a queue-entry is built for it.

The output request is effectively communicated to the Output Intercom Processor by linking the previously built queue-entry to the Output Intercom-Queue. The queue entries are linked according to their assigned priorities with pointers to the first and last linked entries maintained in Communication Region cell .ECQP. If the output message has an originating TASK, the number of assigned and linked queue entries is incremented in cell .TMSG2 of the applicable TASK.

Enabling the Output Intercom Processor

The Output Intercom Processor must be explicitly enabled at the main-level to initiate output processing. The output processor must be enabled whenever a new entry is linked to the Output Intercom-Queue. If the enable does not immediately follow the call to the link routine, the intervening code must be inhibited.

The output processor always returns control when enabled at the main-level. The main-level enable is ignored if the output processor is already executing at the courtesy-call level. The output-processor determines its courtesy-call state by testing communication cell .ECOMO. When this cell is non-zero, the output processor is enabled at the courtesy-call level, so that all currently linked queue entries will be processed.

OUTPUT INTERCOM PROCESSOR

Overview

Once enabled, the output processor retrieves and unlinks the highest priority Output Intercom-Queue entry using communication cell .ECQP. If the queue-entry has an associated TASK, the count of linked queue entries within the TASK is decremented. The contents of the queue-entry are copied and the queue-entry space is released via the .ECOMQ cells.

Starting with the first message segment, the output processor issues an Intercom output request to transfer the segment to TPE. The DCW word count for the segment is obtained from the lower half of the first word of the segment. A courtesy-call is attached to the Intercom I/O so that the output processor will be directly enabled by GCOS when the current Intercom has completed.

When the courtesy-call is paid, the Intercom Status Return is tested to see if the Intercom was successful. If not successful, the Intercom is reissued. Otherwise, the first word of the written segment, the message segment linkage word, is copied and the Output Buffer space assigned to the segment is released. The upper half of the copied linkage word is then examined to determine if there is another segment in the message.

The linkage word holds a pointer to the next segment of the message if it is nonzero; otherwise, the linkage was extracted from the last message segment. If there is another segment, an Intercom output request is issued for it with an appended courtesy-call as before. When this Intercom completes, a check is made to determine if there are any more segments as described above.

Thus the message segments are serially written until the last segment is detected and, consequently, the whole message has been sent. When this occurs, the output processor returns to the Output Intercom-Queue and again fetches the highest priority queue-entry. This procedure is repeated until the queue is empty. At this time, the output processor disables itself by terminating the courtesy-call within an outstanding Intercom I/O.

Notice that the Output Intercom Processor will again be enabled at the main-level when an output request is linked to the Output Intercom-Queue. Furthermore, any requests linked to the queue, while the output processor is enabled, will be retrieved and processed at the courtesy-call level. In the latter case, the main-level enables associated with the newly linked queue-entries would be ignored.

Output Intercom Entry Points

Symbol	Title
II0120	Initiate Output Intercom
II0221	Output Next Message
II0130	Output Next Message Segment
II0131	Reissue Output Intercom
II0140	Output Intercom CC
II0150	Check Intercom I/O Status
II0160	Link Output Intercom-Queue Entry
II0170	Unlink Output Intercom-Queue Entry

FUNCTION:

This routine initiates output Intercom to TPE provided that output to intercom is not already running. Output is sequenced by the priority assigned to each message when its queue-entry was linked to the Output Intercom-Queue. Once started, Output Intercom is driven at the courtesy-call level until the Output Intercom-Queue has been emptied. The Output Intercom routine empties the Output Intercom-Queue from its high priority tail. (It is worth noting that each queue-entry can represent several message segments within the Output Buffer, each requiring its own Intercom I/O to send the complete message to TPE).

ENTRIES:

EP1 - Initiate Output Intercom (II0120).

This entry starts output Intercom with the highest priority entry linked to the Output Intercom-Queue.

No calling parameters.

EP2 - Output Next Message (II0121).

This entry retrieves the highest priority queue-entry to start output of the next message. The entry is used only when the Output Intercom Processor is at the courtesy-call level.

EP3 - Output Next Message Segment (II0130).

This entry builds the necessary DCW to output the next message segment within the queue-entry being serviced.

CP1 - Pointer to next Output Buffer message segment.

EP4 - Reissue Output Intercom (II0131).

This entry reissues the Intercom output request with the same DCW used in the previous issue.

No calling parameters.

RETURNS:

Return is always to the Call+1.

<u>Output Intercom CC</u>

FUNCTION:

This routine checks the Intercom I/O status of the previous GEINOS. If the status indicates that an error has occurred in the previous transmission, the previous Intercom request is reissued to GCOS. Otherwise, the Output Buffer space assigned to the successfully written message segment or message is released, and Intercom I/O is issued for either the next message segment <u>or</u> the first segment of the next highest priority message linked to the Output Intercom-Queue. This routine is courtesy-call driven and continues until the Output Intercom-Queue is empty.

ENTRIES:

EP1 - Output Intercom CC (II0140).

Control passes to this entry point when GCOS services the courtesy-call.

RETURNS:

Return is made to GCOS via a MME GEENDC.

OUTPUT INTERCOM PROCESSOR

Check Intercom I/O Status

FUNCTION:

This routine checks the termination, major status and lost interrupt positions in the first Status Return word of the specified Intercom I/O request.

ENTRIES:

EP1 - Check Intercom I/O Status (II0150).

CP1 - First Intercom I/O Status Return word.

RETURNS:

RR1 - Call+1, Bad Status Return.

The Intercom I/O must be reissued.

RR2 - Call+2, Successful.

Link Output Intercom-Queue Entry

FUNCTION:

This routine links the designated Intercom-Queue entry to the Output Intercom-Queue according to the priority assigned to the queue-entry. The Output Intercom-Queue is ordered by ascending priority values in the forward pointer direction. (See Queues discussion for the queue-entry format). Lastly, the count of linked Intercom-Queue entries, in cell .TMSG2 lower of the indicated TASK (if any), is incremented.

ENTRIES:

EP1 - Link Output Intercom-Queue Entry (II0160).

CP1 - Pointer to queue-entry to be linked.

CP2 - Pointer to associated TASK, if one; otherwise, zero.

RETURNS:

Return is always to Call+1.

Unlink Output Intercom-Queue Entry

FUNCTION:

This routine unlinks the designated queue-entry from the Output Intercom-Queue. The count of linked queue-entries, in cell .TMSG2 lower of the TASK specified by the queue-entry (if any), is decremented by one.

ENTRIES:

EP1 - Unlink Intercom-Queue Entry (II0170).

CP1 - Pointer to queue-entry to be released.

CP2 - Pointer to the associated TASK, if one; otherwise zero.

RETURNS:

Return is always to the Call+1.

EXECUTIVE ERROR MESSAGES

Introduction

Executive error messages are defined by the .EMSG. macro. This macro builds a table of message DCWs along with a table of the variable length messages. Message DCWs are assembled under the ..EXEC location counter, and the actual message texts are assembled under the ..MSG location counter.

Each message is identified by an associated message code, which is used as an index into the list of message DCWs. All references within the Executive to a particular message are made with the error code. This code is normally placed in TASK cell .TERRM upper and is used by the TASK Terminator to determine which message, if any, is to be sent to TPE for a terminating transaction.

There are two special or reserved message codes. Code-1 is used to indicate that no message is to be sent when the TASK is terminated. This code is only used when the TASK belongs to a spawn TASK chain. It is necessary to prevent a message with an End-of-Transaction (EOT) status from being sent to TPE, since this would terminate the transaction's outstanding status within TPE.

Code 0 is used to indicate that only a message header specifying an EOT status is to be sent. This code is used for TASKs that terminate normally and do not belong to a spawn TASK chain. Its effect is to change TPE's processing status of the terminating transaction to 'completed'.

EXECUTIVE ERROR MESSAGES

Error Message Generation

Error messages must be defined in order of their assigned code, i.e., in message code sequence. There can be no missing codes. The .EMSG. macro, with descriptive parameters, is invoked as follows:

```
1      8      16
       .EMSG. Message-Code,

       ETC    Message-Size,

       ETC    (Message Text)
```

where the message size is given in words and must be 9 words or less.

Executive Error Messages

Currently used messages, along with their message codes, are listed below:

Code	Error Message
1	Illegal MME
2	Restricted MME
3	I/O Select Sequence File-Code Address Out of Range
4	I/O Select Sequence Seek DCW Address Out of Range
5	Illegal Seek DCW
6	DCW Data Address Out of Range
7	I/O Select Sequence Status Return Address Out of Range
8	I/O Select Sequence DCW Address Out of Range
9	First DCW a TDCW
10	DCW Address Out of Range
11	Two Successive TDCWs
12	Intercom I/O Status Return Address Out of Range
13	Intercom I/O DCW Address Out of Range
14	Intercom I/O DCW Not an IOTD
15	Intercom I/O DCW Data Address Out of Range
16	Output Initiated Before Input Requested
17	Input Requested After Received

EXECUTIVE ERROR MESSAGES

18	Illegal File-Name
19	Intercom Write DCW Word Count > 128 Words
20	Output Transaction # Not Equal To Input Transaction #
21	Input Requested After Output Initiated
22	TPAP-To-TPAP Message Destination Count Not 1
23	TPAP-To-TPAP Message Keyword Unknown
24	Output Message Segment Destination Counts Disagree
25	Output Message Segment Destination-IDs Disagree
26	Output Message Size Greater Than Executive's Profile
27	Input Message Keyword Unknown
28	Requested TPAP Not Available
29	Fatal Error During Load
30	Fatal Error During Swap-Out
31	Invalid Service Vector
32	Logical-ID Not 3 Characters
33	Message Text Length - Character Count Error
34	KP to KP Message Logical-ID Error
35	Illegal Intercom I/O CMD
36	Output Attempted After EOT Received
37	Output Text Length > Message DCW Count
38	Output Header Incomplete
39	FO-KP Memory Address Fault

40	F1-KP Fault Tag Fault
41	F2-KP Command Fault
42	F3-KP Derail Fault
43	F4-KP Lockup Fault
44	F7-KP Undefined OP Fault
45	F8-KP OP Not Complete Fault
46	F9-KP Overflow/Underflow Fault
47	IO-KP Divide Check Fault
48	<<Reserved>>
49	F6-KP Memory Parity Fault
50	Consecutive GERELC/GEROADs With No Intervening I/O
51	File Control Block Out of Range
52	Aborted By Keyword Processor Code xx
53	Aborted <xx>
54	Invalid MME Parameter
55	Restricted MME Option
56	Resources Exhausted - Try Again
57	GERSTR File-Code Not H*
58	No Overlays Declared In Profile
59	No Overlays Present In Profile
60	GERSTR Call Name Unknown
61	GERSTR Data/Load Origin Too Large For Assembled Memory
62	GERSTR I/O Error
63	Illegal DRL

EXECUTIVE ERROR MESSAGES

64 Restricted DRL

Executive Intercom Entry Points

Symbol	Title
EXM100	Executive Message Intercom
ETX100	Convert & Edit Transaction Number

EXECUTIVE ERROR MESSAGES

Executive Message Intercom

FUNCTION:

This routine builds an output message header using the Transaction number and Source-ID in the designated TASK. The Transaction number is converted to BCD and inserted into the first two words of message text. Output buffer space for the complete message is requested and the header and text are written to buffer.

ENTRIES:

EP1 - Executive Message Intercom (EXM100).

CP1 - L(TASK) to which message applies.

CP2 - Message text DCW. (If zero, only a message header will be built and passed to the output buffer).

CP3 - Message Status (EOS, EOM, or EOT).

RETURNS:

RR1 - Call+1, Sufficient Output Buffer space not available.

RR2 - Call+2, Message successfully written to the Output Buffer. TASK cell .TMSG holds a pointer to the assigned buffer location.

RP1 - Pointer to last assigned buffer cell for this message +1.

Convert & Edit Transaction Number

FUNCTION:

This routine converts the binary transaction number in TASK cell .TNUMB to BCD and edits it into the form #-xxxxxxxxxx, where the number is left justified within the x field.

ENTRIES:

EP1 - Convert & Edit Transaction Number (ETX100).

CP1 - Pointer to TASK whose .TNUMB transaction number is to be edited.

RETURNS:

Return is always to call+1.

RP1 - Edited transaction number.

EXECUTIVE SUPPORTED MME SERVICES

EXECUTIVE SUPPORTED MME SERVICE

Selected MME functions are supported by the Executive in order to mimic GCOS MME processing for the Keyword Processors. An attempt has been made to provide a basic set of functions which are normally required and appropriate to the operating environment provided by the Executive. The design of the Executive's MME Identification/Validation routine and of the existing MME handlers is open-ended so that additional MME functions or MME options can be readily incorporated.

The support provided by the MME handlers varies from completely processing the function internally to basically reissuing the MME to GCOS after preprocessing the MME parameters in order to maintain the integrity of the Executive's operating structure. Of those MMEs currently supported, various restrictions or limitations also apply to the options which are otherwise provided by GCOS.

Control is passed to the requested MME handler by the MME Identification/Validation routine within the GELBAR Fault Handler.

MME Service Symbols

Entry point symbols for the MME services are obtained from the standard GExxxx by replacing the 'GE' with '.G', i.e., .Gxxxx. This is done to avoid duplicate symbols.

Symbols internal to the MME processors have been assigned the forms MMEnnx or Mnnxxx, where nn is the symbolic decimal value of the applicable GExxxx symbol.

MME Service Descriptions

Descriptions of selected MME services are found on the following pages. Detailed descriptions are included only for those MME processors which are somewhat involved or whose incorporation into the Executive requires a multi-faceted interface.

Descriptions for the remainder of the MME processors consist of a function statement along with any necessary supporting information.

EXECUTIVE SUPPORTED MME SERVICES

GEINOS HANDLER

I/O Request Recognition

The MME Identification/Validation routine passes all Keyword Processor MME GEINOS faults to the I/O Request Recognition routine in order to determine whether the request is for device or Intercom I/O. Prior to resolving the I/O type, a check is made to see if a MME GERELC/GEROAD was requested by the Keyword Processor. This is indicated if the B.TRLQ bit flag is on in TASK cell .TFLAG+1. If either a GERELC or GEROAD has been requested, the TASK Status Flag is set off.

The requested I/O type is then determined by examining the File-Code in the Keyword Processor's Select Sequence to see if it is the '::' Intercom File-Code. Control is subsequently passed to the Device or Intercom I/O Handlers as dictated by the designated File-Code.

DEVICE I/O HANDLER

MME GEINOS Preface

FUNCTION:

This routine classifies Keyword Processor MME GEINOS requests as device I/O or Intercom I/O. It also adjusts the 'MME GERELC/GEROAD Requested' TASK Status Flag (B.TRLQ) prior to routing control to the appropriate handler.

ENTRIES:

EP1 - MME GEINOS Preface (.GINOS).

CP1 - Pointer to the Keyword Processor's MME+1 relative to the Executive.

CP2 - Pointer to the TASK requesting I/O.

RETURNS:

RT1 - Intercom I/O Handler

If the request is for Intercom I/O.

RT2 - Device I/O Handler.

If the request is for device I/O.

EXECUTIVE SUPPORTED MME SERVICES

Need for Executive Control

Device I/O is initiated by the Executive for the Keyword Processors, since they cannot be allowed to do their own I/O. The reasons for this restriction are:

(1) All Keyword Processor I/O Select Sequence parameters are relative to the Keyword Processor's GELBAR, not the Executive's LAL, and

(2) The Select Sequence parameters might violate the Keyword Processor's assigned memory boundaries, consequently there would be no protection for any other in-core Keyword Processors or the Executive itself.

To overcome these problems it is necessary for the Executive to preprocess the Select Sequence parameter and reissue the I/O request to GCOS.

Executive I/O Administration Requirements

Several administrative requirements are placed on the Executive in order to supervise and monitor all Keyword Processor device I/O and to maintain the integrity of the operating environment it provides. These requirements are introduced below in terms of their respective needs.

It is imperative that the Executive know which in-core Keyword Processors have outstanding device I/O requests to allow the release or reassignment of a Keyword Processor's assigned core space. In conjunction with this requirement is the need for the Executive to know when each I/O completes so that the outstanding I/O status of the affected Keyword Processor can be adjusted. This control must be explicitly given to the Executive when each I/O completes. Notice that since the Executive initiates I/O for all the Keyword Processors, it cannot implicitly determine when an I/O completes by suspending itself with a relinquish or roadblock.

Lastly, the Executive must be able to determine which Keyword Processor's I/O is completed when it is given control.

Multiple Outstanding I/O Requirements

Prior to describing the actual design of the Executive's Keyword Processor I/O handling, the problems associated with allowing multiple outstanding I/O need to be addressed.

If more than one active I/O request is to be allowed each Keyword Processor (rather than suspending its execution from I/O request to completion), all Keyword Processor Select Sequence parameters and DCWs must be copied to an Executive area. This is necessary to prevent the Keyword Processor from altering the Select Sequence parameters or DCWs while the I/O is active. Copying the Keyword Processor's DCW list would be a problem, since the lists can be of variable length.

In addition to identifying the originating Keyword Processor when an I/O completes, the Executive must also determine which of the identified Keyword Processor's outstanding I/Os it is. This is necessary in order to know which of the copied DCW lists is to be released.

Design Decision & Method

It was decided during the design phase that multiple Keyword Processor I/O would not be initially allowed. Thus all Keyword Processors are suspended from execution when they request device I/O until the I/O has completed. This decision was based primarily on the observation that the average or normal slave I/O processing results in a relinquish being requested after an I/O is initiated. This design decision does not preclude the incorporation of such a feature.

The I/O administrative requirements are effected within the Executive by attaching a courtesy-call to all reissued Keyword Processor I/O requests. The purpose of the courtesy-call is to:

(1) Act as a software interrupt to indicate that some Keyword Processor I/O has completed,

(2) Identify the Keyword Processor that requested the completed I/O, and

(3) Update Keyword Processor status flags and optional I/O related measurements.

The first function is a consequence of the definition of a courtesy-call.

The second function is accomplished by the Courtesy-Call Vector within each TASK. When the Executive reissues the Keyword Processor's I/O, it attaches a courtesy-call address to the Select-Sequence. This address points to cell .TCCV, the TASK associated with the requesting Keyword Processor. The location of the true courtesy-call routine is placed in the Courtesy-Call Vector.

When the courtesy-call is paid, control is passed to the TASK Courtesy-Call Vector. The Courtesy-Call Vector first sets an index register to point to cell .TCCV+1 of the associated TASK; it then passes control to the actual courtesy-call routine. This routine identifies the affected TASK via the previously set index register, prior to performing the third courtesy-call function.

Executive I/O Handling Overview

The Device I/O Handler is functionally divided into the categories of I/O initiation processing and I/O termination processing.

Initiation is triggered by a Keyword Processor's I/O request. Its function is to validate and adjust the

Keyword Processor's Select-Sequence parameters, fill in a skeleton Select-Sequence within the Executive, attach a Courtesy-Call pointer to the applicable TASK Courtesy-Call Vector and to issue the I/O to GCOS. TASK Status Flag B.TDIO in TASK cell .TFLAG is set on at this time to indicate that the Keyword Processor has an outstanding device I/O and is not eligible for processor assignment.

Termination is invoked when the reissued I/O completes. This processing is performed by the Courtesy-Call routine that is attached to the reissued I/O request via the TASK Courtesy-Call Vector.

Device I/O Initiation

The major functions of the Initiation routine are to:

- o Validate the File-Code, DCW and Status Return Select Sequence pointers relative to the Keyword Processor's GELBAR,
- o Ensure that the Status Return is not in the KPA,
- o Verify that the DCW defined data address areas lie within the GELBAR and above the KPA,
- o Relativize the DCW data address(es) within the Keyword Processor to the Executive's LAL,
- o Fill in the Executive's skeleton Select-Sequence with all pointers relative to the Executive's LAL,
- o Insert the address of the TASK Courtesy-Call Vector into the skeleton Select-Sequence and the address of the Courtesy-Call routine in the Courtesy-Call Vector,

- o Set the 'Device I/O in Progress' bit flag on in the TASK, and
- o Reissue the I/O via the skeleton Select-Sequence.

Device I/O Termination

The major functions of the termination routine, which executes at the courtesy-call level, are to:

- o Locate the affected TASK via the index register set by the TASK Courtesy-Call Vector,
- o Re-relativize the DCW data address(es) within the Keyword Processor to the Keyword Processor's LAL,
- o Relativize the Status-Return data address residue to the Keyword Processor's LAL, and
- o Turn off the 'Device I/O in Progress' bit flag in the previously identified TASK.

Keyword Processor Restrictions

There are two restrictions that apply to the Keyword Processors in regard to device I/O. First, courtesy-calls are currently not processed by the Executive. Second, all device I/O commands are allowed except multi-record transfers for a card punch, card reader and printer.

Device I/O Handler

FUNCTION:

This routine serves the Executive in handling all device I/O requested by the Keyword Processors. All I/O Select-Sequence parameters are validated (except the courtesy-call pointer), relative to the Executive, and inserted into a skeleton Select-Sequence used to issue the I/O request to GCOS. The DCW list is validated and made relative to the Executive. A pointer to the Device I/O Courtesy-Call Routine is inserted into the Courtesy-Call Vector (.TCCV) in the TASK, and the location of the TASK Courtesy-Call Vector is placed in the skeleton Select-Sequence as the courtesy-call address. A true I/O request is then issued to GCOS.

EP1 - Device I/O Handler (DIO100).

CP1 - LAL of requesting Keyword Processor.

CP2 - L(TASK) associated with the Keyword Processor.

CP3 - L(MME) relative to the Executive.

RETURNS:

Control passes from this routine as follows:

RT1 - Service Dispatcher's-Queue.

If the device I/O was successfully initiated.

RT2 - Start TASK Abort.

If there was an error in the I/O request.

Validate DCW List

FUNCTION:

This routine follows the DCW list given the pointer to the first DCW from the Keyword Processor's I/O Select-Sequence. For each DCW in the list, its location and the boundaries of the region defined by its data address and word count are checked to ensure that they lie within the Keyword Processor's BAR. Furthermore, the data address region is checked to ensure it does not intersect with the Keyword Prefix Area. The data address is then relativized to the Executive and stored back in the DCW list. Any errors cause the TASK to be marked for abort.

ENTRIES:

EP1 - Validate DCW List (DIO300).

CP1 - L (first Keyword Processor DCW).

CP2 - Pointer to TASK requesting device I/O.

RETURNS:

RR1 - Call+1, Successful validation.

RT1 - Start TASK Abort.

If there was an error in the I/O request.

Device I/O Courtesy-Call

FUNCTION:

This routine re-relativizes each DCW in the Keyword Processor's DCW list to the Keyword Processor's BAR and stores the DCW back into the Keyword Processor's DCW list. The data address residue in the second Status Return word is also relativized to the Keyword Processor's BAR and stored according to the location specified in the Keyword Processor's I/O Select-Sequence. Lastly, the outstanding device I/O bit flag in TASK cell .TFLAG (B.TDIO) is turned off.

ENTRIES:

EP1 - Device I/O Courtesy-Call (DIO400).

Control passes to this entry point when GCOS services the courtesy-call. The location of the affected TASK is recovered from the Index Register set by the Courtesy-Call Vector in cell .TCCV of the TASK.

RETURNS:

Control is returned to GCOS via a MME

INTERCOM

Introduction

All Keyword Processor Intercom is treated as a pseudo Intercom internal to the Executive. That is, a core-to-core transfer either from the Input Intercom Buffer to the Keyword Processor or from the Keyword Processor to the Output Intercom Buffer.

Intercom Status Flags

Several TASK Status Flags have been defined to identify the various Keyword Processor Intercom processing and message states. These flags are used in conjunction with TASK cells .TMSG, the Intercom message descriptor; .TMSG2, the output message descriptor extension: and .TIDCW, the Pseudo Intercom DCW to describe and control each Keyword Processor's input and output Intercom processing.

The processing TASK Status Flags are the 'Building Intercom Output' flag, symbolically denoted B.TBIO, and the 'Intercom Output Complete' flag, denoted by B.TOU flag is set on when the Keyword Processor first requests Intercom output. The 'Intercom Output' flag is set on when an End- of-Transaction (EOT) status is detected in an output Intercom message.

TASK cell .TMSG locates and describes an input or output Intercom message, depending on the Intercom processing status flags, whenever the cell is non-zero. TASK cell .TMSG2 holds a pointer to the last linked Intercom output message segment and the number of linked Output Intercom-Queue entries for this TASK. This count is explained in the Output Processor discussion. Lastly, .TIDCW holds the Keyword Processor's pseudo Intercom DCW when it requests Intercom I/O.

Status Flags & Intercom Output

When a Keyword Processor requests Intercom output, a check is made to see if any output has already been sent. This is done by testing the 'Building Intercom Output' flag in the associated TASK. If the flag is off, .TMSG is checked to see if the input message has been requested (i.e., is zero). If .TMSG is not zero, the input transaction is still in the Input Intercom Buffer; thus, the Keyword Processor is attempting to write Intercom output before requesting the input transaction which it is to process. This is not allowed, so the Keyword Processor is aborted.

If the input transaction has been requested, i.e., .TMSG is zero, the 'Building Intercom Output' flag is turned on to indicate that Intercom output has been initiated by this Keyword Processor. When on, this flag is additionally interpreted to mean that the input transaction has been successfully passed to the Keyword Processor and that TASK cell .TMSG describes an output Intercom message.

For subsequent Intercom output requests by this Keyword Processor, the on status of the 'Building Intercom Output' flag implies that .TMSG could already describe the beginning segments of an output message. Therefore, if .TMSG is not zero, the message being sent by this Keyword Processor request will be linked to the last segment received. The latter segment is located via TASK cell .TMSG2 upper. If .TMSG is zero, the message being sent will be treated as the first or only segment of a new message, depending on the message status.

When an Intercom output message segment specifying an EOT status is received, the 'Intercom Output Complete' flag is set on in the associated TASK. The requesting Keyword Processor is still eligible for processor assignment, even though it has sent its last Intercom Output message. This is done to allow it to perform any necessary wrapup processing.

Status Flags & Intercom Input

When a Keyword Processor requests Intercom input, the 'Building Intercom Output' flag is tested to determine whether TASK cell .TMSG describes an input or output message. If the flag is off, TASK cell .TMSG is checked to see if it has been zeroed (indicating that the input transaction has already been requested). If .TMSG describes the input message (not zero), the message is transferred via pseudo-Intercom to the Keyword Processor. However, if the message has already been requested, the Keyword Processor must be aborted.

If the 'Building Intercom Output' flag is on, it is also necessary to check the 'Intercom Output Complete' flag. If the latter flag is on, the Keyword Processor's transaction processing is considered to be complete. Consequently, the Keyword Processor is terminated normally. If the latter flag is off, the Keyword Processor would be aborted because it is trying to process a new transaction before the Intercom ouput processing of the original transaction has been completed.

Message Status Flags

The remaining Intercom message status flags are the Input and Output Message Types, which are denoted by TASK Status Flags B.TITY and B.TOTY, respectively. The Input Message Type flag specifies that the input transaction lies in the Input Buffer when off, or the Output Buffer, when on. An input message in the Output Buffer means that the message was generated as another Keyword Processor's output.

The Output Message Type flag specifies that the output message is to be sent to TPE, when off, or to another Keyword Processor, when on. This flag is set on whenever an output message has a single destination-ID of '***' (Keyword Processor-to-Keyword Processor communication). This flag is adjusted for each output message as required.

Intercom Request Overview

All Keyword Processor Intercom I/O requests are passed to a common validation routine. This can be done since the format of both the read and write Select-Sequences are identical. The validation routine performs GELBAR boundary checks on the Select-Sequence parameters and the DCW defined data region. If the request is error free the Intercom DCW, always an IOTD, is relativized to the Executive's LAL and stored in cell .TIDCW of the associated TASK.

At this time each Intercom request is classified as a read or write according to the I/O command in the Keyword Processor's Select-Sequence.

Intercom Read Overview

A Keyword Processor Intercom read is always considered to be a request for its input transaction, i.e., the transaction it is to process.

When a read request is received, the Intercom processing status flags are tested, as previously described, to ensure that the request is legitimate. If it is, the 'Input Message Type' flag is examined to determine whether the input message is in the Input or Output Intercom Buffer. If the input message is in the input buffer (the flag is off), a pseudo Intercom routine is called to move the message to the Keyword Processor. TASK cell .TMSG holds the pseudo-Intercom 'from' DCW and cell .TIDCW holds the 'to' DCW for the transfer. When the move is complete, the Input Intercom Buffer space assigned to the requested input transaction is released (see Input/Output Intercom Buffer Maps Discussion), TASK cell .TMSG is zeroed, and a dummy Status Return is built and placed in the Keyword Processor according to its Select-Sequence. The Intercom I/O Handler then returns control to the Dispatcher so that a GELBAR can be reissued to the Keyword Processor.

If the input message is in the output buffer (the message type flag is on), the input message represents Keyword Processor communication. In this case, an input message header is built using the output header of the first message segment in the output buffer. The header is moved to the Keyword Processor, after which the first segment is edited to remove any leading blanks and Logical-ID which may preceed the message keyword. The first segment is then moved to the Keyword Processor. All character counts which precede the message strings that make up the segment, are stripped and, consequently, not transferred as part of the input message. When all message strings in the segment have been moved, the Output Intercom Buffer space assigned to the segment is released.

All message strings of the remaining input message segments are also serially transferred to the Keyword Processor much as the first, until either the latter's DCW word count runs out or the entire input message has been sent. At this time TASK cell .TMSG is cleared and a dummy Status Return is built and stored in the Keyword Processor as dictated by its Select-Sequence. The Intercom I/O Handler returns control to the Dispatcher when all processing has been completed so that a GELBAR can be reissued to the Keyword Processor.

Intercom Write Overview

All Keyword Processor Intercom Output requests are assumed to be output destined for TPE or for Keyword Processor to Keyword Processor communication.

When a write request is received, the Intercom processing status flags are tested, as previously described, to ensure that the request is legitimate. If it is, the 'Building Intercom Output' flag and TASK cell .TMSG are examined to determine if the beginnings of an output message have already been received. If the flag is off, i.e., this is the first Intercom output request, or the flag is on but .TMSG is zero, i.e., this is an Intercom output request to start a new output message, the message segment's destination-ID is checked to see if it is '***'. This destination-ID is used to indicate Keyword Processor-to-Keyword Processor communication. If it is such a message, the destination count is checked to ensure that there is only one destination-ID, and the message keyword is extracted and verified against the Executive's Keywords List, which is located via communications cell .EKEYL. If the destination count or keyword are in error, the Keyword Processor is aborted; otherwise, the processing of this output request rejoins that of all other output messages.

If the 'Building Intercom Output' flag is on and .TMGSG is not zero, one or more message segments have already been written by the Keyword Processor. In this case, the destination count and destination-IDs of the new segment are matched against those in the first message segment to ensure that they are the same. If there is a difference, the Keyword Processor is aborted; otherwise, the processing of the output request rejoins that of all other output messages.

For all output requests, the size of the new message segment is added to the accumulated output message size in TASK cell .TMSG lower. The accumulated size includes all message segment headers and would be zero for the first segment of a message. The result is used to determine if the cumulative size of all active output message segments exceeds the maximum allowable size in .EMXSZ upper. If the size is exceeded, the Keyword Processor is aborted; otherwise, Output Intercom Buffer space is requested for the new segment.

If sufficient buffer space cannot be obtained, the executive will attempt to enlarge the output buffer area by allowing it to annex a block of core from a contiguous free core area. This will be successful unless the contiguous core area is being used.

If sufficient buffer space cannot be obtained, the 'Need Output Intercom Buffer Space' TASK Status Flag (P.TNBS) is turned on and the TASK Service Vector is set so that the processing of the output request can be resumed at a later time when the Intercom I/O Handler is enabled by the Dispatcher's-Queue Service. The handler then returns control to the Dispatcher, since further processing is not possible.

If sufficient buffer space is available, the space is reserved by building a buffer map entry and the segment is then transferred from the Keyword Processor to the Output Intercom Buffer by the pseudo Intercom routine. The actual size of the message segment is stored in the lower half of the first word of the message. This size is required by the Output Intercom Processor, which is described later.

If the new segment is not the first message segment, as determined by TASK cell .TMSG, a pointer to the new segment is placed in the upper of the last linked segment, which is located by cell .TMSG2 upper. Thus all output message segments are linked together via the upper half of the first word of each message segment.

For all output requests, the message segment status is examined. If the status is End-of-Segment, the Intercom I/O Handler returns control to the Dispatcher. If the status is End-of-Transaction, the 'Intercom Output Complete' TASK Status Flag is set on and the message status is changed to an End-of-Message (EOM) Status. Even after this output message has been sent to TPE, the change of message status allows the Executive to also send an Intercom message to TPE, which designates the same transaction number as that assigned to the transaction being processed by the requesting Keyword Processor.

Processing continues for all message segments that specified an EOM status or whose status was changed to EOM. First TASK cell .TMSG upper is cleared, since the last segment pointer is no longer required for this message.

If the message is a 'true' output message for TPE, an attempt to get a free Output Intercom-Queue entry is made. If the attempt is unsuccessful, the TASK is adjusted so that the processing can be later resumed by the Dispatcher's-Queue Service. Otherwise, the queue-entry is filled-in and linked to the queue and the Output Intercom Processor is enabled. (See the Output Intercom Processor discussion for a further explanation.) Lastly, TASK cell .TMSG is cleared for a new message request and control is passed to the Dispatcher.

If the message designated Keyword Processor-to-Keyword Processor communication, an attempt is made to obtain an unassigned TASK, called a spawn TASK, is made. If unsuccessful, the originating TASK is adjusted so that the Dispatcher's-Queue service can re-enable the processing at a later time. Otherwise, a skeleton TASK is built in the free TASK and selected cells are copied from the originating TASK to the spawn TASK.

The keyword is then extracted from the output message so that the designated Keyword Processor Profile can be located and profile specifics inserted into the spawn TASK. At this time, the 'Input Message Type' TASK Status Flag is set on in the spawn TASK and the output message is assigned to it via its .TMSG cell. The spawn TASK is then linked to the Core-Queue and a conditional attempt to allocate core is made depending upon the return taken by the 'Link to Core-Queue' routine. Finally, cell .TMSG of the originating TASK is cleared for a new output message and control is passed to the Dispatcher.

EXECUTIVE SUPPORTED MME SERVICES

Intercom I/O Handler

FUNCTION:

This routine is responsible for reading and writing Intercom messages to and from a requesting Keyword Processor using a MME GEINOS Select-Sequence. For input, the message is passed from the Input or Output Intercom Buffer to the Keyword Processor via pseudo Intercom. After all input is passed, the buffer space assigned to the message is released. For output, an attempt to obtain Output Intercom Buffer space is made and, if successful, the message segment is moved to the output buffer and linked to the previous message segment, if any. If the message segment status is End-of-Message or Transaction, an attempt is made to obtain an Output Intercom-Queue entry and, if successful, the queue-entry is linked to the queue and the Output Intercom Processor is enabled. Keyword Processor-to-Keyword Processor communication is also handled by this routine.

ENTRIES:

Common calling parameters are:

CP1 - Pointer to TASK associated with the Keyword Processor requesting Intercom I/O.

CP2 - L(MME)+1 relative to the Executive.

EP1 - Intercom I/O Handler (II0010).

EP2 - Restart Intercom I/O, Need Output Buffer Space (II0052).

Control passes to this entry point from the Dispatcher's-Queue Service.

EP3 - Restart Intercom I/O, Need Spawn TASK

(II0082).

Control passes to this entry point from the Dispatcher's-Queue Service.

EP4 - Restart Intercom I/O, Need Intercom Q-Entry (II0083).

Control passes to this entry point from the Dispatcher's-Queue Service.

RETURNS:

Control passes from this routine as follows:

RT1 - Reissue GELBAR After Service (DSP302).

Intercom was successfully completed.

RT2 - Service Dispatchers-Queue (DSP102).

Intercom could not be completed.

RT3 - Start Task Abort (TRM100).

Error in request.

RT4 - Start TASK Termination (TRM400).

Intercom read after all Intercom output has been completed.

Validate Intercom I/O

FUNCTION:

This routine validates the Intercom I/O Select-Sequence parameters, relativizes the Keyword Processor's IOTD DCW data address to the Executive and saves the DCW in the associated TASK's .TIDCW cell. Lastly, the IC in TASK cell .TICI is bumped to point to the first word following the Select-Sequence.

ENTRIES:

EP1 - Validate Intercom I/O (II0090).

CP1 - L(TASK) requesting Intercom I/O.

CP2 - L(MME)+1 relative to the Executive.

RETURNS:

Return is made to the calling routine as follows:

RR1 - Call+1, Error in request.

RR2 - Call+2, Successful.

Pseudo Intercom

FUNCTION:

This routine effects a core-to-core data transfer as dictated by the designated 'from' and 'to' DCWs. At the completion of the transfer, the data address and word count of both DCWs are adjusted to reflect the move. The word count used in the transfer is always the smaller of the two DCW word counts, where a word count of zero is taken to be zero unlike a normal DCW.

ENTRIES:

EP1 - Pseudo Intercom (II0100).

CP1 - Pointer to the 'from' DCW.

CP2 - Pointer to the 'to' DCW.

RETURNS:

Return is always made to the Call+1.

RP1 - number of word transferred.

Pseudo Intercom Wrapup

FUNCTION:

This routine builds a pseudo Status Return for a Keyword Processor that has requested Intercom I/O. The result is stored in the Keyword Processor using the Status Return pointer given in its Select-Sequence.

ENTRIES:

EP1 - Pseudo Intercom Wrapup (II0110).

CP1 - Pointer to Status Return in the requesting Keyword Processor.

CP2 - Pointer to the Keyword Processor's DCW used in the pseudo Intercom

CP3 - Pointer to the associated TASK.

RETURNS:

Return is made to the Call+1.

MME GEFCON Handler

FUNCTION:

This routine follows the chain of File Control Blocks (FCB), if any, starting with the FCB pointed to by Q-upper in the designated Keyword Processor's register storage area (.KREG). For each FCB, its range from LOCSYM-7 to LOCSYM and its 'next FCB pointer' in LOCSYM-1 are checked to ensure that they are within the Keyword Processor's BAR. The 'next FCB pointer' is then relativized to the Executive's LAL. The FCB chain is followed until the 'next FCB pointer' is zero or points to the first FCB.

When all FCB pointers in the chain have been relativized to the Executive, a true MME GEFCON is issued. Upon return, all FCB pointers in the chain are re-relativized back to the Keyword Processor's LAL and the GELBAR is reissued.

ENTRIES:

EP1 - MME GEFCON Handler (.GFCON).

CP1 - Pointer to TASK requesting the MME GEFCON.

RETURNS:

RT1 - Reissue GELBAR After Fault (DSP303).

If a valid request.

RT2 - Start TASK Abort (TRM100).

If an error in the request.

MME GEINFO Handler

FUNCTION:

This routine processes Keyword Processor MME GEINFO List Function requests. The SSA Copy Function is not allowed.

The Keyword Processor's Directive List boundaries are checked to ensure that the list lies within the assigned core area. Each directive is then processed with the directive's information address (1 word) and specified option validated.

Currently, only Option 10, Configuration data from .CRFIG, and Option 12, Program Number from SNUMB are implemented. For the latter option, the SNUMB is restricted to $ TRAX.

ENTRIES:

EP1 - MME GEINFO Handler (.GINFO).

CP1 - Pointer Keyword Processor MME+1.

CP2 - Pointer to associated TASK.

RETURNS:

RT1 - Reissue GELBAR After Fault (DSP303).

If a valid request.

RT2 - Start TASK Abort (TRM100).

If an error request.

MME GELAPS Handler

FUNCTION:

Elapsed processor time for the requesting Keyword Processor is copied from the .TLAPS cell of the associated TASK to word 5 of the Keyword Processor's processor register storage area at .KREG in its Keyword Processor Prefix.

ENTRIES:

EP1 - MME GELAPS Handler (.GLAPS).

CP1 - Pointer to the TASK associated with the requesting Keyword Processor.

RETURNS:

Return is always made to the 'Reissue GELBAR After Fault' (DSP303) entry point in the Dispatcher.

MME GERELC/GEROAD Handler

FUNCTION:

This routine handles Keyword Processor MME GERELC or GEROAD faults. The 'MME GERELC/GEROAD Requested' flag (B.TRLQ) setting in the associated TASK is checked first. If off, the flag is set on and control passes to the Dispatcher to reissue the broken GELBAR. If the flag is on, the Keyword Processor is flagged for abort.

The 'MME GERELC/GEROAD Requested' flag is turned off by the MME GEINOS Preface whenever I/O is requested.

ENTRIES:

EP1 - MME GERELC/GEROAD Handler (.GRELC or .GROAD).

Entry is always made from the GELBAR Fault Handler.

CP1 - Pointer to Keyword Processor MME+1.

CP2 - Pointer to associated TASK.

RETURNS:

RT1 - Reissue GELBAR After Fault (DSP303).

If TASK Status Flag B.TRLQ is off.

RT2 - Start TASK Abort (TRM100).

If B.TRLQ is on.

MME GETIME Handler

FUNCTION:

A true MME GETIME is issued for the requesting Keyword Processor with the resulting date and time stored in words 4&5 of the Keyword Procesor's processor register safe-storage area (.KREG).

ENTRIES:

EP1 - MME GETIME Handler (.GTIME).

CP1 - Pointer to requesting TASK.

RETURNS:

Return is always made to the 'Reissue GELBAR After Fault' (DSP303) entry point in the Dispatcher.

EXECUTIVE SUPPORTED MME SERVICES

MME GEWAKE Handler

FUNCTION:

This routine mimics a true MME GEWAKE by suspending the requesting Keyword Processor from execution for the specified time interval. If the time interval is greater than 1 second, the Keyword Processor is marked eligible for swap.

ENTRIES:

Common calling parameters are:

CP1 - Pointer to TASK requesting a GEWAKE.

EP1 - MME GEWAKE Request Handler (.GWAKE).

This entry is used to service the initial GEWAKE request.

EP2 - Awaken Keyword Processor (MME283).

This entry is used to awaken a Keyword Processor when the TASK Alarm Clock (.ETACK) rings.

RETURNS:

RT1 - Start TASK Abort (TRM100).

If an error in the request.

RT2 - Reissue GELBAR after Fault (DSP303).

If zero wait time interval.

RT3 - Service Dispatcher's-Queue (DSP102).

If non-zero wait time interval.

Prototype MME GEROUT Handler

FUNCTION:

This routine processes MME GEROUT requests. Presently, only the Direct Access Output (operation code 3) and the Direct Access Output/Input (operation code 4) functions are handled. The routine is designed to allow easy incorporation of subprocedures for processing other operation codes.

When a GEROUT request is received, address and boundary checks are applied to the GEROUT record pointer, status return word pointer and input buffer pointer, if appropriate. The 'Remote I/O' TASK status bit, B.TRIO, is set true, after which the Remote I/O Supervisor is called to initiate the GEROUT function. Upon return, control is transferred to the Dispatcher since the program requesting the GEROUT cannot be allowed to execute until the I/O is complete.

The GEROUT handler again regains control when it is paid its courtesy-call by the Remote I/O Supervisor. At this time, any GEROUT parameters which were made absolute prior to starting I/O are made relative to the requesting program's LAL, the GEROUT status return is inserted according to the MME GEROUT status return pointer, the 'Remote I/O' TASK status flag is set false and the faulting program's IC is positioned past the MME calling sequence.

ENTRIES:

EP1 - MME GEROUT Handler (.GROUT)

CP1 - Pointer to faulting program's MME+1.

CP2 - Pointer to associated TASK.

EP2 - MME GEROUT Courtesy-Call (MM3050).

CP1 - Terminal Control Block Pointer.

CP2 - GEROUT Status.

CP3 - GEROUT Status Offset.

This offset reflects the bit position, relative to the least significant bit, of the most significant bit set true in the status.

RETURNS:

RT1 - Service Dispatcher's-Queue (DSP,2)

RT2 - Start TASK Abort (TRM,1)

MME GERSTR Handler

FUNCTION:

This routine handles Keyword Processor MME GERSTRs. The Keyword Processor requesting the GERSTR must have the overlay declared in the Keyword Processor Profile.

The overlay entries are scanned for the requested overlay name. If found, the origin and size are determined using the normal GCOS conventions. Then the origin and size are checked to ensure that the overlay will reside within the assigned core for the Keyword Processor. The overlay is then loaded and processing continues according to GCOS conventions.

ENTRIES:

EP1 - MME GERSTR (.GRSTR)

Normal GECOS calling parameters are assumed.

CP1 - Pointer to TASK requesting a GERSTR.

RETURNS:

Returns follow the normal GCOS conventions for 'GERSTR' processing.

RT1 - Reissue GELBAR after Service (DSP302) was successfully completed.

RT2 - Start Task Abort (TRM100) if an error occurred and there was no error handling capability in the GERSTR request.

INITIALIZATION

Introduction

The Executive performs a one-pass initialization of its control functions and the Keyword Processors it is to manage. The Initialization Routine is attached to the main body of the Executive as a linked module, that is by means of a $ LINK control card. This is intentionally done to force all System Library subroutines, which are called by the Executive proper, to be loaded directly behind (above) the Executive's main body and before (below) the Initialization Routine. At the end of the initialization phase, the core space used by the Initialization Routine becomes part of the swap core area. Any error conditions detected during the Executive's initialization are communicated to the console typewriter via error messages. When and if the Executive is ready to begin processing, a start message is sent to the console indicating that the initialization was successful. Due to this procedure, it is suggested that the Executive be initiated from the system console.

Overview

The major functions performed by initialization are as follows:

- Get $TRAX program number
- Initialize the fault vectors
- Initialize the LOAD and SWAP files
- GECALL the Keyword Processors
- Update the Keyword Processor Profiles
- Write Keyword processors to the LOAD-File
- Release excess LOAD-File space
- Set controls for the Core and Swap tables
- Assign input buffer space
- Assign core space
- Assign output buffer space
- Send messages to the console

The GELBAR fault vector within the Slave Prefix

Area (location 19/10) is enabled in the Executive in order for it to process any program errors that are detected in its Keyword Processors. This is necessary because the Keyword Processors are operating within a GELBAR environment, and aborts have to be controlled by the Executive. Recall that the GCOS operating system knows and would abort a program by its program number, which in this case is assigned to the Executive. A precaution was added to the Intercom (F/8) logic in TPE to enhance the security and control of interprogram communication. This control necessitates that the program number, which is to be communicated with, be inserted into the Intercom Select-Sequence I/O command. The program number is requested by SNUMB ($TRAX) using Option 12 of the MME GEINFO. The number is set into the necessary Intercom Select-Sequences with the Executive.

The LOAD-File and SWAP-File are attached to the Executive via the $ FILE control card. The file codes for these files are $L and $S respectively. Both files should be allocated to the fastest device available which has a block size of sixty-four words. The LOAD-File is used to save an initial or clear copy of the Keyword Processors which are GECALLed (see next paragraph) by the Executive. The SWAP-File is used to save the working copies of Keyword Processors and their Keyword Prefix Area when they are suspended from execution. The file attributes of both files are obtained during initialization via MME GEFADDs. If the file does not exist, a MME GEMORE is requested to get a new file. It is suggested that a large number of blocks be allocated to the LOAD-File, so that the space will not be exhausted during initialization. If the space runs out, additional space is requested via a MME GEMORE; however, such a request could be denied. Any excess LOAD-File space will be released via a MME GERELS after all Keyword Processors have been copied onto the file.

The starting address of the Keyword Processor Profile Table is located in cell .EPPFL of the Executive Communication Region. The size of the profile entries are fixed. The maximum input and output message sizes expected by each Keyword Processor are inserted in the profiles at generation time with the .PPFL. macro. Both sizes must be compared to the maximum size allowed by the Executive, as recorded in .EMXSZ, to verify that they are not larger. If either size is larger, the Keyword Processor is rejected and a message is sent to the console to that effect.

The identification of the Keyword Processors to be managed by the Executive are assembled into their respective Keyword Processor Profiles during the generation phase. The Keyword Processor IDs are used within a MME GECALL to load the program into the Executive's high-core The $L seek addresses are placed in the respective profiles for future reference by the Executive. The rationale behind this LOAD-File arrangement is that subsequent I/O requests for the Keyword Processor can be made with a MME GEINOS rather than a MME GECALL. This is beneficial since the GECALL must search a catalog in order to locate the appropriate DCWs to load the named file. On a normal GECALL return, the program size is checked to determine if it will fit the maximum core requirements based on the number of DCWs that can be built in its Keyword Prefix Area. If it cannot be built, it is rejected and a message is sent to the console. If it can, the Keyword Processor's profile is updated to reflect its size and entry address along with the LOAD-File address.

The program size is then checked against the remaining number of blocks (LLBLKs) in the LOAD-File to determine if there is sufficient space. If space is not sufficient, a GEMORE is executed to obtain additional blocks. A dummy TASK is formatted in order to use the Core Allocator's I/O Roll-out routine to write the Keyword Processor to the LOAD-File. The fields that must be set in the TASK are the:

(1) 'New TASK' bit flag (B.TNUT) in .TFLAG+1

(2) $L seek address in .TSWAP upper and the program size in .TSWAP LOWER

(3) GECALL LAL in .TLAL lower.

In addition, a MME GEENDC is placed in TASK cell .TCCV, the Courtesy-Call Vector, since a courtesy-call is not required. To complete this processing phase, a call to the Roll-out routine is made and the Initialization Routine is roadblocked awaiting the completion of the write.

If there is an error in the GECALL or Roll-out, the profile is cleared and a message is sent to the console. When all the profiles have been cycled the excess LOAD-File space is released via a MME GERELS.

The Executive Communication Region is updated with the parameters needed to control the utilization of the Swap-File and the swap core area. For the Swap-File, the number of free 64-word blocks is placed in .ESMAP as the size of the Swap-File and also as the upper address limit of the file. .ESMAP is a four word field that contains the base Swap-File Map entries.

The TPOS Executive then establishes the following:

o Input buffer area
o Available core area
o Output buffer area

The starting address for the input buffer is calculated by determining the address of releasable core. The code in the TPOS Initialization Routine is divided into two areas:

o Code to create the available areas
o Code to perform the other initialization functions

The beginning address of the input buffer area is set at the beginning of 'Code to perform the other initialization functions'. The amount of space needed is calculated as follows:

```
  Recommended Input Intercom Buffer Space (See Site
      (Reference)
+ 1.5 * maximum input message size
+ the number of words to round to the beginning of the
  next 1024-word block
  ----------------------------------------------------
  Total used for the input buffer space
```

```
                   ______________________________
                  |          INPUT               |
                  |          BUFFER              |
                  .                              .
                  .______________________________.
LAST ENTRY        |            MBZ               |
                  |                              |
                  |PTR TO- - - - -|- - - - - - - |
                  |.EIMAP         |     RESV     |
INPUT BUFFER      |__________THRESH______________|
THRESHOLD         |                              |
                  |NUMBER WORDS   | NUMBER OF    |
                  |AT INIT.       | WORDS NOW    |
                  |               |              |
                  |PTR TO - - - - | UNUSED - - - |
                  |START BUFFER   |              |
                  |______________________________|
                  |          INPUT               |
                  |          BUFFER              |
                  |         THRESHOLD            |
                  |______________________________|
                  |        AVAILABLE             |
                  |          CORE                |
                  .                              .
                  .                              .
                  |       AVAILABLE              |
                  |__________CORE________________|
                  |          OUTPUT              |
                  |          BUFFER              |
                  .        THRESHOLD             .
                  |______________________________|
                  |                              |
                  |          OUTPUT              |
                  |          BUFFER              |
                  .                              .
                  .                              .
                  |______________________________|
LAST ENTRY        |            MBZ               |
                  |- - - - - - - - - - - - - - - |
                  |PTR TO          |             |
                  |.EOMAP          |             |
THRESHOLD         |__________THRESH______________|
ENTRY             |- - - - - - - - - - - - - - - |
                  |NUMBER WORDS    | NUMBER OF   |
                  |AT INIT.        | WORDS NOW   |
                  |------------------------------|
                  |PTR TO          | UNUSED      |
                  |START BUFFER    |             |
                  |________________|_____________|
```

The ending address of the Output Intercom Buffer is the UAL (found in word 31 (decimal) of the slave prefix area). The beginning address of the Output Intercom Buffer is calculated as follows:

```
  The UAL
- the recommended size of the Output Intercom Buffer
  ------------------------------------------------
  Subtotal
- The number of words necessary to round
  down to the next even 1024-word block
  ------------------------------------------------
  Beginning address
```

A threshold area is created for both the Input and Output Intercom areas. The threshold is currently set to 1.5 times the maximum input/output message size.

The size of the threshold area is subtracted from the input buffer upper address limit. The initialization of the input buffer and input buffer queue is completed.

The core area is then reserved and the core-queue and maximum core size are initialized.

The Output Intercom queue, the output threshold area and the Output Intercom area are then initialized.

At the completion of the Executive's Initialization, a start message is sent to the operator's console.

Initialization Console Messages

Several messages can be issued to the console in response to various conditions that can be encountered during initialization. A list of current messages follows. The SNUMB is that assigned to the Executive.

```
SNUMB**RESTART TRAX

SNUMB*INSUFFICIENT LOAD-FILE FOR TPOS EXEC

SNUMB*NO SWAP FILE FOR TPOS EXEC

SNUMB*K.P.S.xxx INPUT BUFFER TOO LARGE

SNUMB*K.P.S.xxx OUTPUT BUFFER TOO LARGE

SNUMB*K.P.S.xxx TOO LARGE

SNUMB*K.P.S.xxx GECALL ERROR

SNUMB*K.P.S.xxx LOAD-FILE ERROR

SNUMB*NO ACTIVE K.P. FOR TPOS EXEC

SNUMB*TPOS EXECUTIVE INITIATED
```

Where xxx is the Keyword Processor's ID.

MISSION
of
Rome Air Development Center

RADC plans and conducts research, exploratory and advanced development programs in command, control, and communications (C^3) activities, and in the C^3 areas of information sciences and intelligence. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.

ATE
LMED
-7